uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática y
Administración de Empresas
2018-2019

Trabajo Fin de Grado

"Estudio de herramientas de despliegue continuo de aplicaciones, y sus ventajas competitivas en un mundo marcado por la agilidad"

Carlos Sánchez Gómez
Tutores
Román López-Cortijo García
Covadonga Gijón
Colmenarejo, 2019

RESUMEN

El presente documento recoge el estudio realizado sobre el *Modelo Ágil* y continuo de despliegue de aplicaciones, y la utilización de la *metodología Agile* en las empresas, y su efecto sobre la eficiencia de estas. Dicho estudio analiza las ventajas del *Modelo Ágil* sobre el *Modelo Tradicional* de despliegue de aplicaciones, expone una guía didáctica sobre como crear, configurar y utilizar un entorno de despliegue continuo, y realiza un análisis cualitativo y cuantitativo sobre la influencia positiva que puede tener la utilización de las técnicas ágiles de trabajo en la empresa.

En el trabajo, se contraponen las dos metodologías predominantes en el despliegue de aplicaciones, analizando las cuatro fases del flujo de trabajo de cada una, y viendo sus diferencias en cada una de ellas, concluyendo que el *Modelo Ágil* tiene más ventajas y encaja mejor en el desarrollo de software. Tras esta conclusión, se describe, de forma exhaustiva, el proceso de creación y configuración de un entorno de despliegue de aplicaciones aplicando el *Modelo Ágil*, su utilización (con un caso práctico), y las herramientas que se han usado en el proceso.

Posteriormente, se realizará un análisis cualitativo sobre la utilización de la *metodología Agile* en la empresa, sus características principales, y ejemplos reales de su aplicación, y un *análisis DEA* (cuantitativo) sobre la eficiencia de las empresas que utilizan esta metodología, para concluir el efecto positivo que incluir esta metodología en la forma de trabajo de una empresa puede tener tanto sobre la eficiencia de la misma, como, por ejemplo, sobre los beneficios y satisfacción de los empleados.

Palabras clave: Ágil; Integración continua; Despliegue; DEA; Metodología

ABSTRACT

This document includes the study carried out on the *Agile Model* of continuous application deployment, and the use of the *Agile methodology* in companies, and its effect on their efficiency. This study analyzes the advantages of the *Agile Model* over the *Traditional Model* of application deployment, presents a didactic guide on how to create, configure and use a continuous deployment environment, and performs a qualitative and quantitative analysis on the positive influence that the utilization of agile work techniques in a company can have.

In this research, the two predominant methodologies in the deployment of applications are compared, analyzing the four phases of the workflow of each one, and describing their differences in each of them, concluding that the *Agile Model* has more advantages and fits better in software development. After this, it is explained, in an exhaustive way, the process of creating and configuring an application deployment environment applying the *Agile Model*, its use (with a practical case), and the tools that have been used in the process.

Subsequently, a qualitative analysis will be carried out on the use of the *Agile methodology* in a company, its main characteristics, and real examples of its application, and a *DEA* (quantitative) *analysis* on the efficiency of the companies that will use this methodology, to conclude the positive effect that including this methodology in the way a company works can have both in its efficiency and, for example, in the benefits and satisfaction of employees.

Keywords: Agile; Continuous integration; Deployment; DEA; Methodology

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

I. INTRODUCCIÓN

La gestión e implementación de proyectos de desarrollo de software no son técnicas novedosas, para ver su origen hay que remontarse a la aparición de los primeros proyectos de software, es decir, de la ingeniería del software, disciplina que comienza en la década de los 80, ante la necesidad de contar, al igual que en las demás ramas de la ingeniería, con una que agrupe todas las técnicas, estándares y metodología para el diseño y la construcción de productos de software (Pantaleo & Rinaudo, 2015).

Dentro del desarrollo del software, uno de los campos más importantes, y el que se tratará en este trabajo, es el **despliegue de las aplicaciones** que se implementan en los proyectos de software. Para realizar esta actividad, las empresas siguen modelos que se caracterizan por tener unas pautas a seguir para el despliegue de aplicaciones. A día de hoy, existen dos de estos modelos, totalmente opuestos: el ***Modelo Tradicional***, el que se ha utilizado siempre, y el ***Modelo Ágil***, más novedoso, que ha empezado a utilizarse en los últimos años (Pantaleo & Rinaudo, 2015).

Este trabajo se centrará en la última de estas, con dos objetivos establecidos: demostrar que el *Modelo Ágil* es claramente la mejor opción a elegir para el desarrollo y despliegue de software, y exponer un caso práctico y didáctico en el que se realice dicha actividad, y que pueda servir de guía para todo aquel que desee instalar, configurar y utilizar un entorno de despliegue continuo de aplicaciones.

Además de ser muy importantes en el mundo del software, las técnicas ágiles, comúnmente conocidas con el nombre de ***Metodología Agile***, han empezado a aparecer en numerosas empresas de diversos sectores (financiero, telecomunicaciones, industrial…), cambiando por completo la forma de trabajo de estas. Las empresas están en constante evolución, y deben innovar para no quedarse atrás con respecto a sus competidores. Es por esto que la *Metodología Agile* se está abriendo un hueco, y existe suficiente evidencia como para afirmar que tiene un efecto positivo en ellas.

Es por ello que, en este trabajo, se analizará también este supuesto efecto positivo que la incorporación de la *Metodología Agile* puede tener sobre una empresa que decide implantarla. Por una parte, de forma cualitativa, se estudiará cuales son todas las características de esta nueva forma de trabajo, y como cambia la organización y actividad de una empresa, con sus respectivos beneficios. Por otra, de forma cuantitativa, se

realizará un análisis con datos reales sobre diez entidades financieras que han incluido esta metodología de trabajo en los últimos años, a fin de extraer conclusiones verídicas sobre el supuesto efecto positivo de dicha metodología.

Este trabajo, en resumen, servirá para toda persona, grupo o entidad, que desee implementar técnicas ágiles (tanto en software como en forma de trabajo), de forma que, por una parte, puedan comprobar con datos verídicos los beneficios de las mismas, y por otra, guiarles en el camino a dicha implementación.

Con respecto a la estructura del trabajo, se ha divido en dos partes claramente diferenciadas. En la **primera parte**, se estudiará el despliegue continuo de aplicaciones, en el que primero se analizarán los **dos modelos existentes de despliegue de aplicaciones**, comparándolos de forma que se pueda extraer una conclusión sobre cual es mejor, y, por la otra, se explicará, de forma didáctica, el proceso de **creación, configuración y uso de un entorno de despliegue continuo de aplicaciones**, analizando previamente cada una de las herramientas que se utilizarán en el proceso.

En la **segunda parte**, se estudiará la importancia de las metodologías ágiles en las empresas y su efecto en la eficiencia de estas, realizando primero un **repaso de la literatura previa** existente sobre este tema. A continuación, se describirá tanto el origen como el significado de todos y cada uno de los **datos** que se han utilizado para el análisis de eficiencia, posteriormente la **metodología** utilizada, con una breve introducción sobre el tipo de análisis utilizado, la explicación matemática del modelo utilizado para resolver el problema planteado y la descripción del caso práctico a resolver, y, por último, los **resultados** obtenidos tras la realización del análisis.

Tras estas dos partes del trabajo, se realizará un resumen de las **conclusiones** que se han podido extraer tras la finalización del trabajo, junto a las limitaciones que se han ido encontrado durante su realización, y los pasos a seguir o posibles trabajos futuros y posteriores al mismo.

Toda la documentación utilizada en la realización de este trabajo se puede encontrar en la **bibliografía** (libros, informes anuales y páginas web citadas) y en los **anexos** (tablas de datos utilizadas para el análisis, y tablas de resultados obtenidas tras el análisis).

II. DESPLIEGUE CONTINUO DE APLICACIONES

A. ESTADO DEL ARTE

En este primer apartado, se analizarán, por un lado, tanto el ***Modelo Tradicional*** de despliegue de aplicaciones, como el ***Modelo Ágil*** de despliegue de aplicaciones, recorriendo el flujo de trabajo propio de un proyecto de software para ver cuales son sus diferencias más significativas, y, por otro lado, se describirán todos y cada uno de los aspectos más importantes del *Modelo Ágil* de despliegue de aplicaciones.

1. Modelo Tradicional vs. Modelo Ágil de despliegue de aplicaciones

1.1 Modelo Tradicional de despliegue de aplicaciones

El primer modelo de despliegue de aplicaciones que se explicará será el ***Modelo Tradicional,*** caracterizado, sobre todo, por la leve o nula automatización de todos sus procesos. Para la explicación de las características de este modelo, se disgregará en cuatro partes, que representan el flujo de trabajo en cualquier proyecto de ingeniería de software, y que serán fundamentales a la hora de realizar una comparación posterior entre ellos: equipos, construcción de código, pruebas y despliegue. En la figura 1 se puede ver, de forma gráfica, un resumen del flujo de trabajo en el *Modelo Tradicional*, con todas las partes que se describirán a continuación.

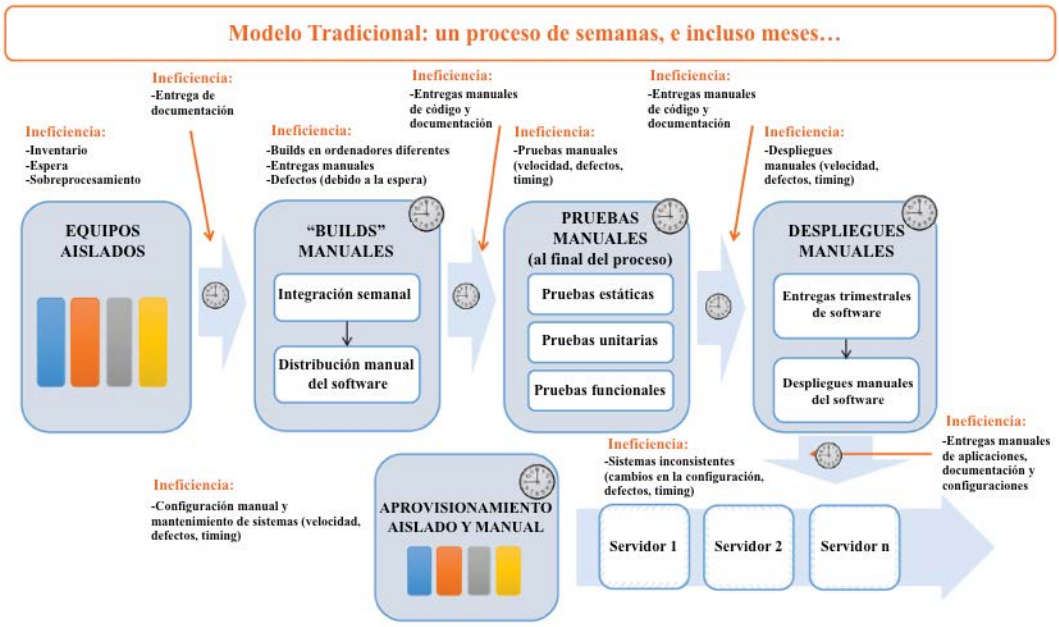


Figura 1. Modelo Tradicional de despliegue de aplicaciones (Farenhorst, 2016)

Los **equipos** en el *Modelo Tradicional* se caracterizan, sobre todo, por ser aislados. Cada equipo de un proyecto de software trabaja independientemente en la tarea que le ha sido asignada, y están todos dirigidos por la figura del Project Manager, el cual se encarga de coordinar a los diferentes equipos, para enfocar el trabajo de todos ellos a la consecución del proyecto. Estos equipos son equipos especializados: cada uno se encarga de una tarea concreta, agrupando en cada equipo a las personas que mejor encajan para realizar esa tarea, de acuerdo a sus competencias. Debido a la separación y aislamiento en las tareas del proyecto, los equipos que siguen este modelo de desarrollo suelen ser bastante grandes (Hito Master DAP, 2017).

Respecto a la **construcción de código** (la cual se entiende por el proceso mediante el cual un desarrollador programa aquella tarea que se le ha asignado), cada equipo implementa sólo la tarea que se le ha asignado. En ambos modelos (*Tradicional* y *Ágil*), la construcción del código tiene un factor común: existe un código fuente o código principal (el conocido como *mainline*), que es el que los desarrolladores irán modificando hasta llegar a completarlo con todas las funcionalidades requeridas por el cliente. Cuando un desarrollador quiere introducir un cambio en dicho código, realiza una copia en local del mismo, y se dispone a realizar su tarea. En el *Modelo Tradicional*, el desarrollador sigue la técnica conocida como *Integración Frecuente,* que consiste en la integración (entendiendo por integración la introducción de los cambios que el desarrollador quiere insertar en el *mainline*) de su copia en local del código (la que él mismo realizo para introducir cambios) de forma "frecuente", esto es, día a día (o incluso semana a semana), en el *mainline.* En el modelo tradicional, esta integración, al igual que la distribución de software que se realice sobre los equipos, es manual (Pressman, 2010).

Aquí surge un conflicto a la hora de integrar el código, pues durante el tiempo en que un desarrollador está cambiando el *mainline* en local, otro puede haberlo modificado, con lo que la copia que se había realizado en local va reflejando cada vez menos el *mainline* activo. Cuanto más tiempo pase desde que se realiza la copia hasta que se modifica el *mainline*, más probabilidad habrá de que este haya sido modificado por otro desarrollador, y, por lo tanto, más costoso será conseguir que los cambios compilen con el *mainline*, pues todos los cambios que se han ido produciendo en el *mainline* desde que realizamos la copia, no se han tenido en cuenta a la hora de programar (Pressman, 2010).

Así, aparece uno de los problemas que más quebraderos de cabeza puede producir a la hora de realizar un proyecto de software, el conocido como *integration hell*, que consiste básicamente en que el tiempo que el desarrollador pierde en conseguir que su código con los cambios a introducir compile con el *mainline*, sea mayor que el tiempo que ha consumido en implementarlos; un problema que, como veremos más adelante, también se da en el *Modelo Ágil* (Patel, 2014).

La fase de **pruebas**, en la que se realizan todas las pruebas necesarias para comprobar el correcto funcionamiento del código (**estáticas**, que no necesitan ejecutar código y que comprueban la calidad del código en función de unos estándares; **unitarias**, que comprueban que partes aisladas del código funcionan sin errores; **funcionales**, que comprueban que el resultado de ejecutar código es el correcto), se caracteriza porque se realizan de forma manual; las pruebas las realizan los desarrolladores, y las implementan una vez acabado el código (Pressman, 2010).

El **despliegue** o **fase de producción**, la cual consiste en entregar al cliente el producto, con todas las funcionalidades requeridas operativas, y formarle para que consiga realizar un correcto uso de este. En el *Modelo Tradicional*, dicha entrega se realiza manualmente, y siempre una vez terminado el código, y pasadas las pruebas descritas anteriormente (Pressman, 2010).

1.2 Modelo Ágil de despliegue de aplicaciones

El siguiente modelo de despliegue de aplicaciones que se explicará será el ***Modelo Ágil,*** caracterizado, sobre todo, automatización de todos sus procesos. Para la explicación de las características de este modelo, al igual que para la del *Modelo Tradicional*, se disgregará en las cuatro partes correspondientes al flujo de trabajo en cualquier proyecto de ingeniería de software. En la figura 2 se puede ver, de forma gráfica, un resumen del flujo de trabajo en el *Modelo Ágil*, con todas las partes que se describirán a continuación.

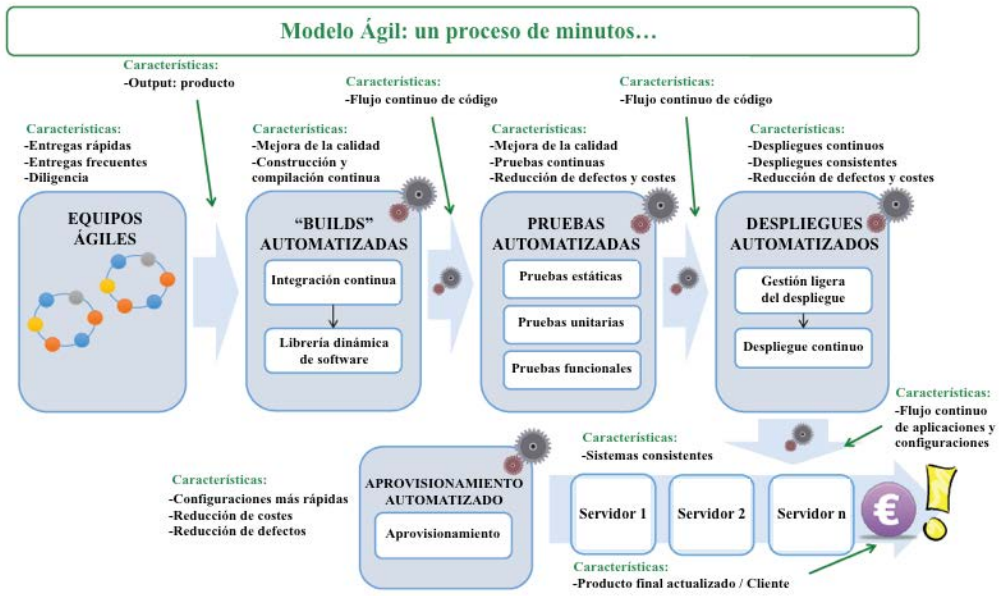


Figura 2. Modelo Ágil de despliegue de aplicaciones (Farenhorst, 2016)

En el *Modelo Ágil*, los **equipos** son multidisciplinares, es decir, cada uno de los miembros del equipo tiene conocimientos sobre varios campos diferentes, sin llegar a estar especializado en ninguno en concreto. Así, todos deben ser capaces de realizar las tareas del resto de compañeros, y pueden ayudarse y aprender unos de otros. Los equipos deben estar bien integrados entre ellos, ser capaces de cooperar, comunicarse y darse apoyo para mantener activo el flujo de trabajo. Es importante también en estos equipos que las metas sean comunes y no individuales: el objetivo es único, para todos los miembros del proyecto, y es entregarle al cliente el producto en el *deadline* marcado. Esto reafirma la importancia de la integración y apoyo entre equipos mencionada anteriormente. Cabe destacar que estos equipos son autogestionados, es decir, los miembros del equipo son los que se organizan, desde el reparto de tareas hasta las fechas que se marcan para la realización de estas. Debido a las características que se acaban de describir respecto a estos equipos, se entiende que los mismos estén formados por un pequeño número de personas, pues es mejor la organización y la distribución de tareas cuando se realiza por pocas personas (Hito Master DAP, 2017).

La **construcción de código** en el *Modelo Ágil* se torna más compleja. Se mantiene el mismo flujo de trabajo que en el *Modelo Tradicional*, mediante copias en local del *mainline*, pero se introduce una nueva técnica, la que se conoce como *Integración Continua* (a partir de ahora, IC), una de las técnicas más conocidas del *Modelo Ágil* de despliegue de aplicaciones. Dicha técnica trata de agilizar y optimizar el proceso de

integración de las tareas que componen un proyecto. Para entender cómo trata de conseguir esto la IC, se estudiará el flujo de trabajo propio de esta técnica, partiendo desde el momento en que se quiere introducir un cambio en el *mainline*, hasta que se realiza. La primera parte del flujo es la misma que para el *Modelo Tradicional*, el desarrollador realiza una copia en local del *mainline,* sobre la cual trabajará. Esto se hace porque es inviable que varios equipos trabajen sobre el mismo código (Pressman, 2010).

Como ocurría en el *Modelo Tradicional*, cuanto más tiempo tarde un desarrollador en integrar los cambios que realice sobre el código (en terminología ágil, realizar un *commit*), más probable será que otro desarrollador haya realizado cambios sobre el *mainline*, y más tiempo le llevará integrar sus cambios en el *mainline,* pues tendrá que adaptarlos a los cambios que hayan hecho el resto de desarrolladores durante el tiempo que ha estado programándolos (el conocido como *integration hell).* La IC trata de resolver este problema mediante un proceso de integración muy exhaustivo, en el que se disgregan los cambios que el desarrollador tiene que aplicar al código en cambios lo más atómicos posibles, para así realizar integraciones cada menos tiempo, es decir, que la IC conlleva realizar más de una integración al día (Patel, 2014).

La IC consiste básicamente en esto, pero además existen una serie de buenas prácticas que pueden ayudar más a que la IC sea realmente útil. Compilar y ejecutar el *mainline* una vez realizado un *commit* sería una de ellas, pues es posible que, al subir el código, se pueda haber olvidado alguna librería o archivo necesario para su correcto funcionamiento, y crear un error en el *mainline* que, al igual que el resto de errores que puedan aparecer en el código, cuanto más pase el tiempo, más difícil será de detectar. Esta compilación (también conocida como *build*) puede ser manual (llevada a cabo por el mismo desarrollador que sube el código) o automática, si se utiliza un servidor de Integración Continua, el cual, según el desarrollador realiza el *commit*, envía una notificación al desarrollador transmitiéndole el resultado de la compilación (Pressman, 2010).

Las **pruebas** en el modelo ágil se caracterizan, sobre todo, por su automatización. El desarrollador debe implementar toda la batería de pruebas (estáticas, unitarias y funcionales, al igual que en el modelo tradicional) incluso antes de empezar con la tarea que le ha sido asignada. Una vez implementados, se incluyen en el servidor de Integración Continua, el cual, al igual que realiza la compilación automática sobre el *mainline* al realizar un *commit*, también es capaz de correr todas las pruebas sobre el *commit* que se

está intentando realizar, informando también al desarrollador sobre el resultado de dichas pruebas. Es importante realizar las pruebas después de la compilación, pues que el código compile no quiere decir que funcione como debería. Así, se consigue detectar el error nada más introducirlo. Dado que es muy probable que cada desarrollador trabaje en un entorno de desarrollo diferente al de producción, es también frecuente realizar las pruebas en un entorno que imite al máximo las condiciones del entorno de producción, para evitar así cualquier error que pueda surgir con respecto a trabajar en diferentes entornos (Pressman, 2010).

Por último, se describirá la **fase de producción** del *Modelo Ágil*, que, al igual que las pruebas, está automatizada. Al igual que aparece la IC en la construcción de código, en la fase de producción encontramos la *Entrega Continua*, que consiste en dividir el proyecto en fases en las que, al final de cada una, el proyecto esté listo para su despliegue (es decir, para pasar a producción, al cliente). Sin embargo, que esté listo para producción no significa que se mande a producción; de eso se encarga el *Despliegue Continuo*, que automatiza el proceso de despliegue para, ahora sí, mandar el proyecto a producción. Por lo tanto, se deduce que, sin *Entrega Continua*, no podemos tener *Despliegue Continuo*. Romper el proyecto en varias fases de entrega implica que el cliente pueda ir viendo el desarrollo del proyecto, lo que puede ayudar tanto a detectar posibles errores que no se vieron durante la fase de construcción, como a mejorar el proyecto con nuevas ideas que puedan surgir a raíz de, por ejemplo, nuevas necesidades del cliente (Pressman, 2010).

1.3 Comparativa entre ambos modelos

Una vez descritas las características y particularidades de ambos modelos, se realizará una comparación entre ellos, siguiendo la misma estructura que se utilizó para describirlos, y que permita sacar una conclusión acerca de cuál de los dos modelos es mejor.

La primera diferencia entre ambos modelos se encuentra en la estructuración y organización de los **equipos** del proyecto. Al ser más antiguo, el *Modelo Tradicional* adopta un sistema de gestión del liderazgo más autocrático, es decir, la responsabilidad de liderar el proyecto recae sobre una persona (el Project Manager), que se encarga de dirigir y supervisar el proyecto, nombrando también a jefes en cada uno de los equipos, lo cual le permite llevar un seguimiento más exhaustivo de cada uno de los bloques del

proyecto. En el *Modelo Ágil*, esta organización es totalmente opuesta. La responsabilidad de liderar el proyecto recae directamente sobre los equipos, ellos mismos se organizan sus tareas, sus *deadlines*…

Recogiendo las características de ambos modelos respecto a este punto, no se puede afirmar que ninguna de las dos formas de organización sea mejor que la otra; todo depende del tamaño del proyecto, siendo más apropiado el modelo tradicional para proyectos de grandes dimensiones, y el ágil para proyectos más pequeños.

En la **construcción del código** es donde se encuentra una de las diferencias más notorias entre ambos modelos. Como se explica anteriormente, el flujo de construcción de código funciona igual en ambos modelos, pero la diferencia radica en la utilización de la *Integración Frecuente* en el *Modelo Tradicional*, y de la *Integración Continua* en el *Modelo Ágil*. Ambas son técnicas de integración, pero en la primera se realizan integraciones día a día, mientras que en la segunda se puede encontrar un proceso de integración más frecuente, en el que se realizarán varias de ellas al día. A priori, puede parecer que la CI sea mejor que la *Integración Frecuente*, pues, aunque no consiga solventar al cien por cien el problema del *Integration Hell* (por muchos *commit* que realicemos, siempre puede darse el caso de que un desarrollador realice un cambio en el *mainline* antes que nosotros), si es verdad que consigue reducir bastante la aparición de dicho conflicto. Ahora bien, el problema de la IC aparece en el momento en el que hay que implantar dicha técnica en un equipo/proyecto cuya cultura se basa en el *Modelo Tradicional*. En función de la capacidad de adaptación a los cambios, y la aversión a los mismos que dicho equipo pueda tener, será mejor o peor intentar implantar dicha técnica.

Los beneficios de esta técnica son evidentes: se reducen los problemas que plantea la integración, se realiza una supervisión más exhaustiva del proyecto que tiene consecuencia la reducción de errores, y la mayor facilidad para obtener una imagen real del avance del proyecto (el *mainline* está en constante cambio, reflejando las modificaciones de los desarrolladores a tiempo casi real). A estos beneficios habrá que oponer, como ya ha visto, la dificultad que puede suponer implantar la técnica de la IC en un equipo, además de los costes que ello puede suponer (implantación de nuevas herramientas, formaciones…).

En la fase de **pruebas** se pueden encontrar también diferencias muy significativas. Hay que oponer la realización manual de las pruebas en el *Modelo Tradicional*, frente a la automatización de estas en el *Modelo Ágil*. Vivimos en un mundo en el que la

automatización de procesos es cada vez más común, y el desarrollo de software no iba a ser menos. Es obvio que cualquier proceso que se automatice va a suponer menos tiempo al desarrollador, que podrá invertirlo en otras tareas. Además, automatizar este proceso elimina la posibilidad de errores humanos en una tarea tan importante como comprobar que el producto que se va a entregar al cliente funciona correctamente y siguiendo los requerimientos que el mismo pidió. La otra gran diferencia es la frecuencia de aplicación de dichas pruebas. Con un proceso automático, se posibilita la realización de estas pruebas con mayor frecuencia que si hubiera que realizarlas manualmente, por eso encontramos que en el *Modelo Tradicional* se realizan una vez se termina el producto, mientras que en el *Modelo Ágil* se realizan pruebas de forma continua, lo que facilita también la detección de errores con mayor antelación (los cuáles serán más fáciles de corregir cuanto antes se detecten, dado que si no los corregimos a tiempo, estaremos programando sobre un código que no funciona como debería).

Si bien es cierto que puede parecer que las pruebas automatizadas tienen más beneficios, hay que tener en cuenta que una prueba manual también puede ser útil cuando queremos probar una funcionalidad muy específica del proyecto, o cuando queremos detectar un *bug* que las pruebas automáticas hayan pasado por alto (dado que dichas pruebas automáticas han tenido que ser programadas previamente por un humano, y pueden haber pasado por alto ciertas comprobaciones). Una buena programación de las pruebas automatizadas puede conseguir un elevado porcentaje de éxito, pero nunca llegará al cien por cien, y ahí es donde debemos complementarlas con las pruebas manuales.

Por último, estudiaremos las principales diferencias entre el despliegue o **fase de producción** de ambos modelos. Se enfrentará el despliegue manual del producto en el *Modelo Tradicional*, frente al despliegue continuo del *Modelo Ágil*. En este punto, la diferencia la marca el hecho de que, en el *Modelo Tradicional*, el cliente no puede ver el producto hasta que se realiza el despliegue final (hasta que está terminado), mientras que, en el *Modelo Ágil*, el cliente, como ya se ha visto antes, llega a ser considerado como "uno más del equipo", pudiendo disfrutar, gracias al despliegue continuo, de una vista real del desarrollo del producto (el producto está constantemente en producción).

Esta diferencia se refleja directamente en el retorno de la inversión que se obtiene del proyecto. En el *Modelo Tradicional*, al no tener el proyecto en producción hasta haber finalizado el mismo, no se empieza a darle valor (a obtener ingresos por el mismo) hasta

la finalización de este, mientras que, en el *Modelo Ágil*, al estar el proyecto en un proceso de despliegue continuo, se empieza a obtener valor desde el primer despliegue, cerca del comienzo del proyecto, y se va incrementando gradualmente según se van realizando nuevos despliegues. En la figura 3 se puede apreciar esta diferencia gráficamente.

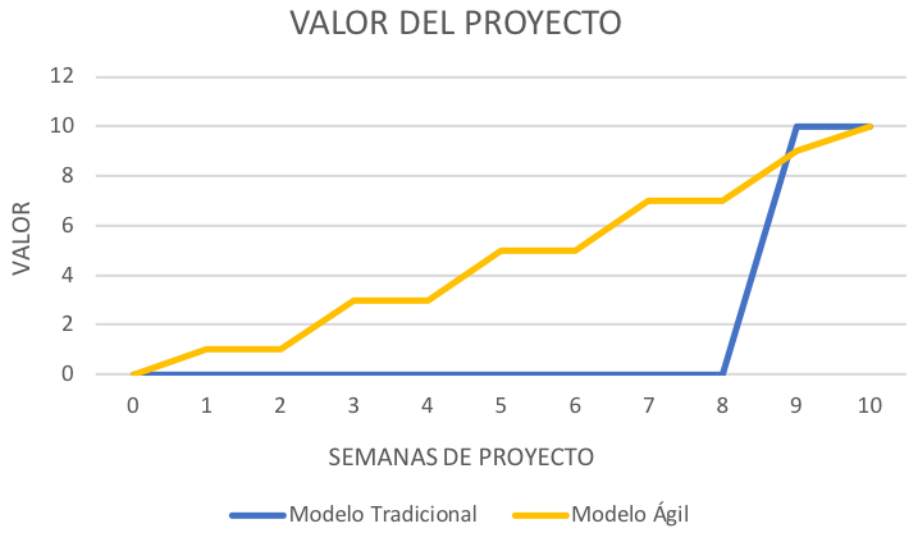


Figura 3. Comparación entre el incremento del valor del proyecto entre el modelo tradicional y el modelo ágil (Elaboración propia)

Una vez enfrentadas las principales características de modelos, será más fácil obtener una conclusión acerca de cuál de los dos modelos es mejor. A priori, puede parecer que el *Modelo Ágil*, dado que es el más novedoso y el que más innovaciones introduce, será el que se imponga sobre el otro. Sin embargo, si se desglosan ambos modelos en bloques (tal y como hemos hecho en la comparativa previa, resumida de forma más visual en la Figura 4), se puede apreciar que no es tan evidente como puede parecer a simple vista.

	Modelo Tradicional de despliegue	Modelo Ágil de despliegue
Equipos	-Dirección autocrática del proyecto (Project Manager) -Proyectos de gran tamaño	-Equipos autogestionados -Proyectos de tamaño pequeño
Construcción de código	*-Integración Frecuente* -No se elimina el *integration hell*	*-Integración Continua* -Se consigue reducir el *integration hell* -Se añade el coste de implantar una técnica nueva en el proyecto
Pruebas	-Pruebas manuales -Al final del proyecto	-Pruebas automáticas -Continuamente, en cada integración
Fase de producción	-Despliegue manual del proyecto, al final de este -Aparición de valor en el proyecto al final del desarrollo de este	-Despliegue automático del proyecto, de forma continua, durante el desarrollo de este -Aumento incremental del valor del proyecto durante el desarrollo de este

Figura 4. Tabla comparativa entre el Modelo Tradicional y el Modelo Ágil (Elaboración propia)

En campos como la estructuración y organización de **equipos**, por ejemplo, se puede concluir que no es que haya un modelo u otro mejor, sino que cada uno se adapta mejor a distintos tipos de proyecto. No tendría sentido instaurar un sistema autocrático en un proyecto formado por solo 10 personas, ni sería eficiente permitir la autogestión de este en un proyecto formado por un gran número de equipos, pues la comunicación entre ellos sería muy complicada.

También se puede apreciar cierta posibilidad de coexistencia de ambos modelos en las **pruebas**, pues, aunque bien es cierto que una ejecución continua y automática de

las pruebas (*Modelo Ágil*) consigue una reducción bastante alta de los errores, no se puede llegar a prescindir de las pruebas manuales (*Modelo Tradicional*) las cuales, programadas directamente por el desarrollador, pueden llegar a detectar fallos que una prueba automatizada puede haber pasado por alto.

Sin embargo, en los dos bloques restantes, sí existe evidencia suficiente de que el *Modelo Ágil* incorpora novedades que pueden convertirlo en un modelo más eficiente que el *Modelo Tradicional*. Estos bloques son la **construcción de código** y la **fase de producción**, en las que el proceso de automatización y aumento de la frecuencia, tanto de integración como de despliegue, consiguen mejorar tanto el proceso de desarrollo como el de producción, tal y como se ha visto en la comparación previa (aunque hay que tener en cuenta el coste que pueda suponer implantar estas nuevas técnicas en un equipo de desarrollo).

Tras este análisis, la **conclusión** que se puede extraer es que, a pesar de que el *Modelo Ágil* no consigue solventar al cien por cien todos los problemas que presenta el *Modelo Tradicional*, sí lo hace en gran medida, lo que le convierte en mejor candidato a la hora de elegir un modelo a seguir, si bien es importante tener en cuenta ciertas prácticas del *Modelo Tradicional* (como la estructuración del equipo o las pruebas a realizar) en función del tipo de proyecto que se vaya a desarrollar, que puede ayudarnos más incluso a aumentar la eficiencia, valor, y consistencia del mismo.

2. Aspectos clave del *Modelo Ágil* despliegue de aplicaciones

A partir de aquí, el trabajo se centrará en el *Modelo Ágil* de despliegue de aplicaciones, el cual se describirá en profundidad, analizando cada una de las técnicas utilizadas en las distintas fases del proceso de entrega y despliegue del código, las cuales se muestran de forma gráfica en la figura 5, explicadas brevemente a continuación:

- **Organización ágil**: Una organización ágil adopta principios ágiles y ajustados combinados con equipos autónomos y multidisciplinares (Farenhorst, 2016).
- ***Build* automatizada:** la compilación automatizada se define como el proceso automatizado para transformar los cambios de código (*commits*) automáticamente en productos o aplicaciones listas para el despliegue, listos para la validación en distintos entornos, de forma coherente (Farenhorst, 2016).

- **Pruebas automatizadas:** Las pruebas automatizadas implican la ejecución automatizada de pruebas o scripts. Los tipos de pruebas que más se usan son las pruebas estáticas, pruebas unitarias y pruebas funcionales (Farenhorst, 2016).
- **Despliegue automatizado:** Se define como el proceso para desplegar automáticamente productos o aplicaciones listas para el despliegue en los entornos de despliegue correspondientes. Este proceso incluye tanto los pasos necesarios para desplegar el producto o aplicación en el servidor de producción, como la configuración previa de dicho servidor (Farenhorst, 2016).
- **Aprovisionamiento automatizado:** Este proceso es el que debe garantizar que tanto los componentes de los entornos de prueba, como los componentes de red y de servidor, se puedan crear a petición del usuario mediante un proceso totalmente automatizado (Farenhorst, 2016).
- **Arquitectura**: La arquitectura (o arquitecturas) definidas en todos los niveles, tales como empresa, negocio o aplicación, determinan tanto la capacidad de optimizar el proceso de entrega de software de los equipos, como la complejidad de adoptar cambios en el código (Farenhorst, 2016).

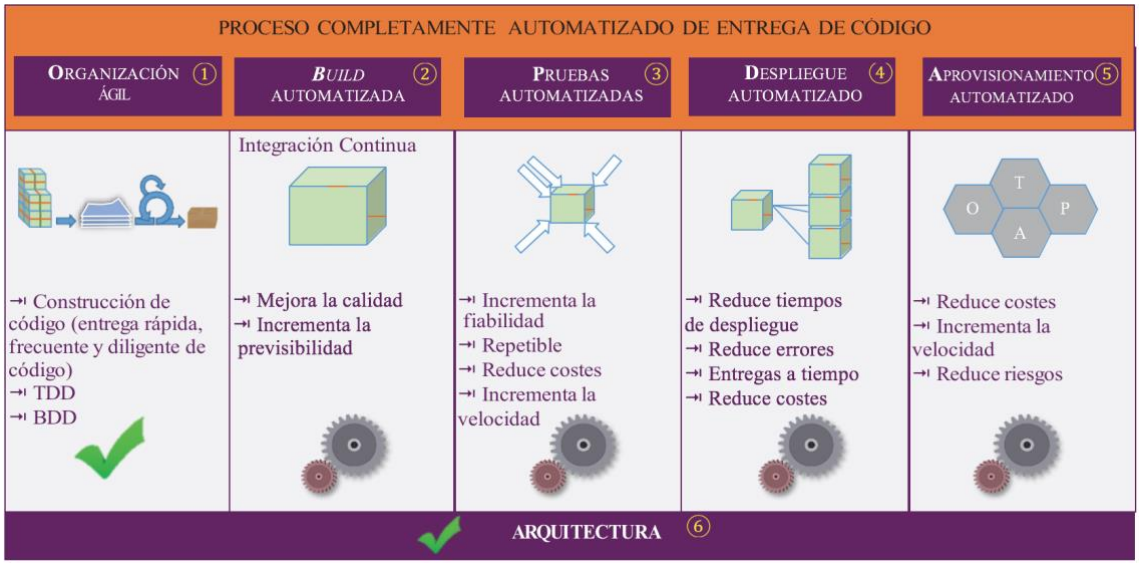


Figura 5. Proceso de entrega de código en el Modelo Ágil (Farenhorst, 2016)

A continuación, se explicarán las diferentes técnicas utilizadas durante el desarrollo de las fases descritas en el *Modelo Ágil* de despliegue de aplicaciones.

2.1 Organización ágil

2.1.1 Construcción de código: Integración continua

La *Integración Continua* (a partir de ahora IC), es una de las técnicas imprescindibles en el desarrollo ágil de software. Consiste en la actualización del *mainline* cada vez que uno de los desarrolladores realice un pequeño cambio, esto es, varias veces al día (Fowler, 2006). Con esto se consiguen dos beneficios fundamentales:

-Se reduce el tiempo que se tarda en detectar un error, pues este solo puede encontrarse en el fragmento de código nuevo que hemos introducido (al realizar varias integraciones al día, este fragmento de código será más pequeño que si, por ejemplo, realizáramos una al día) (ThoughtWorks Inc., 2019).

-Se reduce la posibilidad de que otro desarrollador introduzca un cambio mientras nosotros estamos implementando cambios, y, por lo tanto, evita tener que adaptar el código a los cambios que otro ha introducido, para que este compile (evitamos el *integration hell*) (Fowler, 2006).

2.1.2 Test-Driven Development (TDD)

Esta técnica (*Desarrollo Dirigido por Pruebas* en español, a partir de ahora TDD), de diseño e implementación de software, es una práctica de programación que consiste básicamente en realizar los siguientes pasos: implementar pruebas, escribir el código que pase dichas pruebas, y por último refactorizar dicho código (Blé, 2010). A continuación, se detallarán dichos pasos más a fondo:

- **Implementar pruebas:** El primer paso que debe seguir un desarrollador a la hora de utilizar esta técnica, es implementar la prueba que refleje el requerimiento (uno de ellos, habrá que realizar este paso por cada requerimiento que haya) que el cliente le ha solicitado, es decir, la prueba que compruebe que el producto hace lo que el cliente ha pedido (Blé, 2010). Al ejecutar esta prueba, tiene que fallar, puesto que no debería haber nada implementado sobre este requerimiento (si no fallase, tendríamos que rehacer la prueba).

- **Escribir el código:** Una vez tenemos la prueba implementada, debemos escribir el código que hace que dicha prueba tenga un resultado correcto al ser ejecutada. Este código debe ser lo más simple posible, es decir, hay que escribir la menor cantidad de código que haga que dicha prueba funcione (Ambler, 2003).

- **Refactorizar**: El proceso de refactorización consiste en reescribir código de forma que el mismo se simplifique, quitando redundancias o código innecesario, pero manteniendo la funcionalidad (el comportamiento) del mismo intacta (Beck & Fowler, 1999). Este sería el último de los pasos de la programación TDD, al final del cual ya se tendría un fragmento de código que representa uno de los requerimientos del cliente sobre el producto.

Estos tres pasos se deben repetir de forma cíclica, comenzando por el primer requerimiento a implementar, y terminando con el último de ellos, después del cual finalizaría el proceso de desarrollo del código, pues ya estaría terminado. En la figura 6 se puede observar el flujo de trabajo del TDD anteriormente descrito, de forma gráfica.

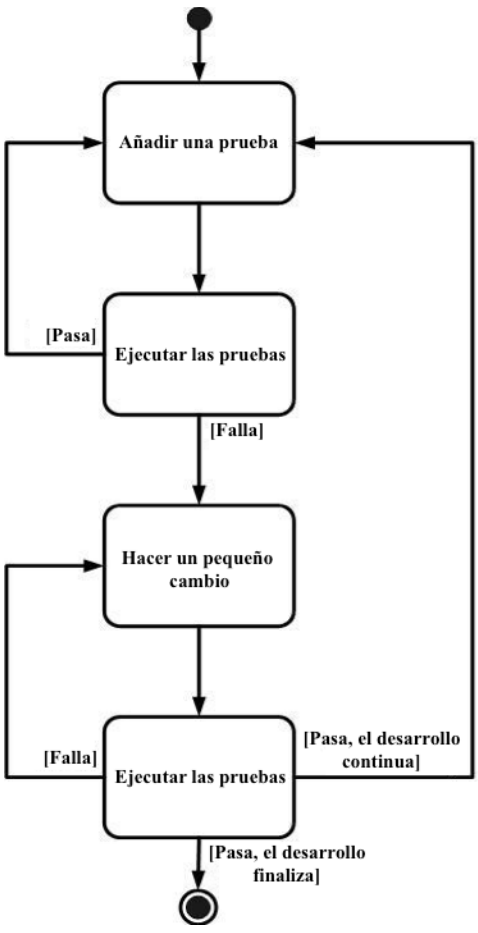


Figura 6. Flujo de trabajo de la técnica TDD (Ambler, 2003)

2.1.3 Behaviour-Driven Development (BDD)

El *Behaviour-Driven Development* (*Desarrollo Dirigido por Comportamiento* en español, a partir de ahora BDD) es una técnica que surgió como evolución del TDD, para dar solución a los problemas que dicha técnica pudiera ocasionar (North, 2006). El objetivo principal de esta técnica es que tanto la parte de desarrollo como la de negocio del proyecto, usen un lenguaje de programación (*Gherkin*) muy cercano al natural, que les permita entender todas las partes del dicho proyecto. El flujo de trabajo del BDD se basa esencialmente en la creación de pruebas de aceptación (que comprueban que la funcionalidad implementada funciona bien). Consta de tres fases (Garzás, 2014):

- **Historias de usuario:** Para crear las pruebas de aceptación, hay que empezar por las historias de usuario, tarjetas que reúnen las características de cada uno de los requerimientos del cliente.

- **Escribir escenarios:** Consiste en definir los criterios de aceptación para dichas historias de usuario, esto es, definir los posibles escenarios para cada una de ellas ("dadas una serie de características, si ocurre un evento, debe pasar lo siguiente").

- **Automatizar las pruebas:** Una vez definidos los requerimientos y los criterios, ya tenemos las pruebas, que debemos automatizar para poder probar nuestro código de forma continua.

2.2. Build automátizada: Integración continua

La Integración continua (IC) también implica un proceso de compilación (también conocido como *build*) con el *mainline*, el cual puede ser manual (realizado por el propio desarrollador) o automático (el más popular), que consiste en la utilización de un *servidor de IC* que, al recibir las diferentes integraciones de los desarrolladores, realiza una compilación automática sobre el *mainline*, informando al desarrollador del resultado (Fowler, 2006).

Es también importante destacar las buenas prácticas de la IC (a tener en cuenta tanto en el proceso de construcción de código, como en el de compilación), para un correcto uso de ella (Humble & Farley, 2011; Fowler, 2006; ThoughtWorks Inc., 2019):

-Tener un único repositorio raíz, donde esté todo lo necesario para que el *mainline* compile y funcione correctamente.

-Automatizar la compilación del *mainline*.

-Automatizar las pruebas del *mainline*, pues una correcta compilación de este no nos asegura que funcione correctamente.

-Realizar *commit* al menos una vez al día, para reducir posibles errores y pérdidas de tiempo al compilar (también ayuda a motivar al desarrollador al marcarle objetivos más a corto plazo).

-Compilar el *mainline* después de un *commit*, para comprobar que se ha integrado correctamente.

-Arreglar errores según se detectan; cuanto más tiempo pase, más difícil será arreglarlo.

-Hacer las pruebas en un entorno idéntico al de producción, para no pasar por alto errores que puedan surgir del uso de diferentes entornos de programación.

-Comunicación con el equipo; a pesar de estar todo muy automatizado, el equipo debe estar al tanto de todos los cambios que se van realizando, quién los ha hecho, y cómo los ha hecho, para facilitar aun más el proceso de integración.

2.3 Automatización de pruebas

La automatización de pruebas (de la que ya se habló en la explicación del *Modelo Ágil*) no es otra cosa que la programación de las pruebas (estáticas, unitarias y funcionales) para que se ejecuten una vez se haya realizado la compilación del *mainline* (también automática), pues, como se ha explicado anteriormente, una compilación correcta no quiere decir que el código funcione como debería (Pressman, 2010).

También es importante que el entorno en el que se ejecuten las pruebas sea lo más parecido posible al entorno de producción, para evitar errores de cara al proceso de producción (Pressman, 2010).

2.4 Despliegue automatizado

2.4.1 Entrega Continua (EC)

La *Entrega Continua* (a partir de ahora EC), es otra de las técnicas fundamentales del *Modelo Ágil* de despliegue de aplicaciones. Esta técnica consiste en que el software que estamos implementando, se encuentre constantemente disponible para pasar a producción, es decir, que reúna las condiciones necesarias para ser usado por el cliente (Humble, 2017). Esta técnica funciona como puente entre la mencionada IC, y el *Despliegue Continuo* (fase de producción), el cual se explicará más adelante; una vez el código se ha integrado con el *mainline* (IC), se prepara para dejarlo disponible para pasar a producción (Garzás, 2012). De aquí se deduce que la EC es diferente de la fase de producción, pues dejar algo listo para producir no implica producirlo. Así, se consigue dejar el producto preparado para cuando los encargados de negocio del proyecto decidan lanzarlo al cliente (Garzás, 2012). La EC se puede (y debe) automatizar para sacarle el máximo rendimiento, de manera que cada vez que se realice una compilación del *mainline*, se realicen una serie de pruebas automáticas que aseguren el correcto funcionamiento de este, de cara a ser usado por el cliente, eliminando así el factor humano en un paso del desarrollo de software en el que es muy fácil cometer un error, y puede suponer un retraso importante en el despliegue final del producto (TechTarget, 2017).

La técnica de la EC tiene bastantes beneficios en el desarrollo de un proyecto software, de los cuales destacan (Humble & Farley, 2011; Humble, Continous Delivery, 2017):

- Se reduce mucho el tiempo que se tarda en lanzar un producto al mercado, pues está en todo momento preparado para su despliegue.

- Se aumenta la calidad del producto, puesto que el desarrollador puede invertir el tiempo que tendría que usar en dejar el producto listo para su despliegue, en realizar otro tipo de pruebas más específicas y detalladas, que permitan pulir aun más el producto.

- Al dejar el producto listo para despliegue, se puede obtener *feedback* muy útil del cliente que permita mejorar el producto, adaptarlo mejor a sus necesidades o corregir errores funcionales que se hayan pasado por alto. Todo esto hace que el producto final acabe siendo mejor.

2.4.2 Despliegue Continuo (DC)

Si la EC consistía en dejar el producto listo para su despliegue, el *Despliegue Continuo* (a partir de ahora DC) consiste en la automatización del proceso de producción, es decir, aprovechando la EC que deja el producto listo para ser utilizado por el cliente, el DC lo pone en producción (Garzás, 2012).

Esta técnica tiene una serie de beneficios que la distinguen del tradicional despliegue manual de software (Agile Alliance, 2019), como la obtención de retorno de la inversión gradualmente a lo largo del proceso de desarrollo (y no al final, dado que se despliega varias veces durante el proyecto), o la obtención de *feedback* de los usuarios que están usando el producto recién desplegado.

2.5 Aprovisionamiento automatizado

En lugar de aprovisionar y administrar componentes de entorno manualmente, como un servidor de aplicaciones, en el *Modelo Ágil* de despliegue de aplicaciones, se pueden adquirir nuevos componentes de entorno a petición del usuario, totalmente automatizados. No hace falta ninguna configuración manual, instalación, configuración ni mantenimiento. El aprovisionamiento automatizado se define como la entrega y el mantenimiento totalmente automatizados de los componentes del entorno de aplicaciones (Farenhorst, 2016).

Los beneficios de esta nueva forma de aprovisionamiento son evidentes:

- La creación de un servidor de aplicaciones pasa de necesitar días para su configuración y entrega (*Modelo Tradicional)*, a estar listo en cuestión de horas (*Modelo Ágil)* (Farenhorst, 2016).
- Es mucho más sencilla la creación de estos servidores o máquinas, pues las herramientas que existen para utilizar esta forma de aprovisionamiento cuentan con innumerables configuraciones e imágenes de máquinas, entre las cuales se puede elegir (*Modelo Ágil)*, en lugar de tener que configurar cada una de las características de la maquina que queremos crear (*Modelo Tradicional)* (Farenhorst, 2016).
- Las maquinas o servidores creados con el aprovisionamiento automático se almacenan *online*, en "la nube", donde tendremos toda la memoria que necesitemos (al contrario que de forma local, donde la memoria es limitada), y

además el proveedor de este servicio de aprovisionamiento nos garantiza la seguridad de nuestros servidores (un ejemplo de estos proveedores sería la empresa *Google*) (Farenhorst, 2016).

B. ANÁLISIS DE HERRAMIENTAS DE INTEGRACIÓN CONTINUA

A continuación, se describirán todas y cada una de las herramientas de integración continua que se han utilizado en el trabajo para crear un entorno de integración y despliegue continuo, y desarrollar un proyecto con sobre dicho entorno. En la figura 7, de forma esquemática, se resume el flujo de trabajo de un proyecto software siguiendo el *Modelo Ágil* de despliegue de aplicaciones, indicando que herramienta se ha utilizado en cada parte del proceso.

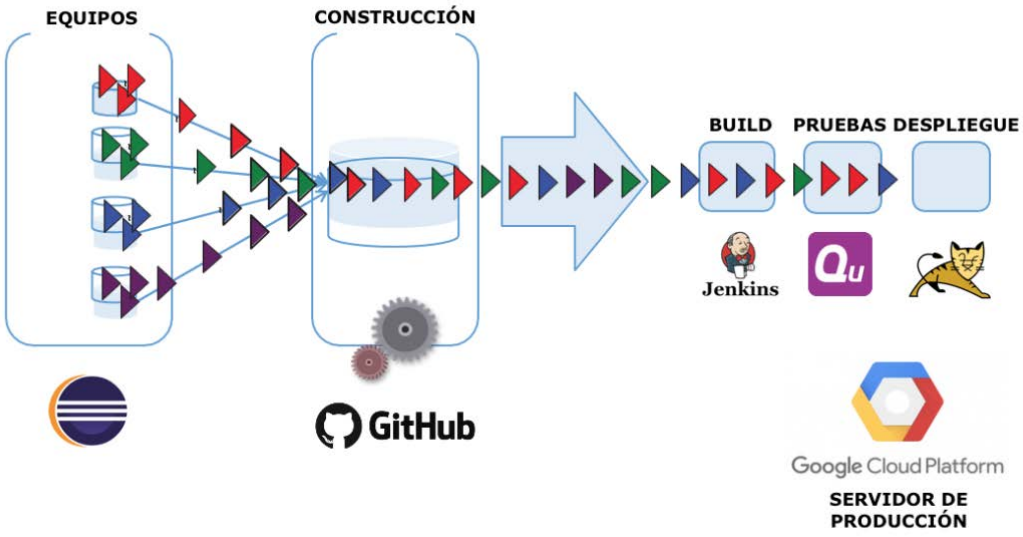


Figura 7. Flujo de trabajo de la técnica TDD (Ambler, 2003)

1. Google Cloud Platform

Google Cloud Platform (Google, 2019a) es un servicio en línea ofrecido por la empresa Google, mediante el cual el usuario puede utilizar todas y cada una las herramientas online ofrecidas por esta plataforma, encuadradas en multitud de categorías, entre las cuales se pueden encontrar, entre otras: *Networking, Big Data, Developer Tools, Management Tools* y *Computing* (García, 2018).

En este trabajo, se utilizarán herramientas incluidas en la última de las categorías mencionadas, concretamente el llamado ***Compute Engine***. Mediante esta herramienta, el usuario puede **crear multitud de servidores o instancias online**, dotándolas del sistema operativo y, por ende, del lenguaje que ellos mismos deseen (Google ofrece una amplia lista de imágenes de sistemas operativos, pudiendo el usuario instalar cualquier otra imagen que haya descargado de otra fuente) (García, 2018). El funcionamiento de esta herramienta es similar al del resto de herramientas de creación de máquinas virtuales (VirtualBox, VMWare,…), con la única diferencia de alojar dicha máquina en la *nube*. Por lo demás, tanto la configuración de la máquina (sistema operativo, memoria de almacenamiento, memoria RAM, puertos,…) como su lanzamiento, es el mismo.

De todas las herramientas que se utilizarán en el desarrollo de este trabajo, esta es la única que conlleva un coste. Sin embargo, Google ofrece, gratuitamente, un total de 267€ anuales cuando se crea la cuenta en *Google Cloud* para utilizar en ella. Si se tiene en cuenta que los costes de crear un servidor online y tenerlo siempre en funcionamiento (con las funcionalidades que vamos a utilizar en este proyecto) han sumado un total de unos 36 € en unos cuatro meses (ver figura 8), en un año entero se acumularan unos costes de unos 108€, con lo que todavía quedarían unos 159€ para gastar en las diversas herramientas que nos ofrece *Google Cloud.* En resumidas cuentas, **la utilización de este servicio no supondrá coste alguno**.

Figura 8. Resumen de gastos en Google Cloud durante cuatro meses (Elaboración propia)

La alternativa a utilizar esta herramienta sería la utilización de servidores almacenados de forma local, con la creación de máquinas virtuales. Sin embargo, utilizar un servicio de infraestructura de servidores en la *nube* como el que ofrece *Google Cloud*, proporciona una serie de **ventajas** exclusivas, como la **seguridad** que aporta mantener los servidores en línea bajo la supervisión de una de las empresas informáticas más

reconocidas del mundo como es Google o la **liberación** de los desarrolladores de las tareas de gestión de la infraestructura (llevada a cabo por Google), para pasar a centrarse directamente en la implementación y despliegue de aplicaciones (Google, 2019b).

2. Apache HTTP Server

Más conocido como *Apache*, y desarrollado por la llamada *The Apache Software Foundation*, es un **servidor web** basado en *Linux*, cuya principal función es otorgar al usuario la posibilidad de lanzar su contenido a la web. (Ubuntu, 2014). Fue lanzado por primera vez en el año 1995, y desde entonces ha estado en constante revisión y mejora (Bustos, 2019).

Realmente, *Apache* no es un servidor web en sí, sino un software que permite conectar el servidor del cliente con los navegadores de internet, para visualizar el contenido que se desea desplegar en el navegador elegido, funcionando tanto en *Unix* como en *Microsoft* (Bustos, 2019).

La utilización de *Apache* es totalmente gratuita, tanto para particulares como para empresas. Es la herramienta **más utilizada** de estas características, la que más soporte ofrece, y la que permite utilizar la herramienta *Tomcat*, que se verá en el siguiente punto, esencial para el despliegue de aplicaciones Java en la web (Bustos, 2019), ventajas que determinaron la elección de esta herramienta para realizar la función de servidor web.

3. Tomcat

También de código abierto, *Tomcat* es un software desarrollado también por *The Apache Software Foundation*, lanzado unos años más tarde que *Apache* (concretamente en 1999). Su función es muy parecida a la de *Apache*: servir como puente al usuario para lanzar su contenido / aplicaciones a la web. Sin embargo, lo que diferencia a *Tomcat* de *Apache,* es que permite el lanzamiento de aplicaciones Java a la web (The Apache Software Foundation, 2019).

Al igual que *Apache*, es gratuita para cualquier tipo de usuario, y es ideal para trabajar con servidores en la *nube*, pues será más fácil controlar el tráfico web, y ajustar los recursos destinados por el servidor al despliegue de aplicaciones en función de dicho tráfico (ARSYS, 2017).

4. Github

Github, basado en el sistema de control de versiones *Git*, no es otra cosa que una plataforma online de creación y almacenamiento de repositorios (almacenes de código) de carácter público (cualquier usuario puede descargarlos y/o modificarlos) o privado, a elección del usuario que crea dicho repositorio (Miró, 2017). Fue creado en 2008, alcanzando en solo 7 años los 9 millones de usuarios, y es utilizado por importantes instituciones como Facebook, o el Ayuntamiento de Madrid, entre muchas otras (Siles, 2015). En el año 2018 fue adquirido por unos 7.5 billones de dólares por *Microsoft,* lo que demuestra el potencial que tiene esta herramienta (Giret, 2018).

También de uso gratuito, su sencillez y su funcionalidad de control de versiones, convierten a *Github* en el perfecto aliado para construir y utilizar un entorno de integración continua. Con un simple *click*, el desarrollador puede sincronizar su progreso en el código con el repositorio donde esté almacenado el código fuente, para que cualquier miembro del equipo pueda acceder a los cambios, dejando un resgistro de todas y cada una de las actualizaciones que se han realizado en el código, haciendo más fácil la detección y corrección de errores en el código (Miró, 2017).

Otra de las ventajas de *Github* es la seguridad que ofrece este tipo de repositorios: con el aval de una gran empresa como *Microsoft,* que garantiza la protección y almacenamiento del código, podemos prescindir de almacenar el código en servidores locales, en los que se corren riesgos como el borrado o robo del código (Finley, 2012).

5. Jenkins

Jenkins es una de las herramientas más conocidas y utilizadas en el mundo *agile*, proporciona al usuario un servidor de integración y despliegue continuos, totalmente gratuito, con multitud de *plugins* que permiten a la aplicación integrarse con casi cualquier herramienta o software que utilicemos para programar (Jenkins, 2019). Nació en 2011, inicialmente llamado Hudson, de la mano de Kohsuke Kawaguchi (Garzás, JavierGarzas.com, 2014).

La funcionalidad de *Jenkins* es esencial en el desarrollo del proyecto que nos ocupa: servirá de puente entre el repositorio de *Github* (donde se irán almacenando las versiones del código) y la parte de producción (despliegue del código en el servidor web), esto es, *Jenkins* detectará automáticamente cualquier actualización del código en *Github*,

lo compilará y comprobará que no haya errores, y lo mandará directamente a producción. Posee una interfaz gráfica muy simple e intuitiva que permite a cualquier tipo de usuario (en términos de conocimientos informáticos) entender como funciona, y darle un uso óptimo a la herramienta.

6. QUnit

QUnit es, más que una herramienta, un *framework* de pruebas de código exclusivo para Javascript, razón por la cual se ha elegido para realizar las pruebas del código que se ha desarrollado en este trabajo. Su funcionamiento es muy sencillo: sólo hay que añadir unas etiquetas *html* en el código de nuestra página, y crear un script (que se cargará también en el *html* de la página) en el que se incluirán las pruebas. Al ejecutar el *html*, las pruebas se ejecutarán también, mostrando el resultado de estas en la parte baja de la página (The JQuery foundation, 2019).

La facilidad para aprender a realizar este tipo de pruebas debido a su sencillez, y la no necesidad de descargar ninguna aplicación, herramienta o entorno para su uso la convierten en el *framework* perfecto para la ejecución de cualquier tipo de prueba (Benítez, 2011).

7. Eclipse

Eclipse es un entorno de desarrollo integrado (comúnmente llamado IDE, del inglés *Integrated Development Enviroment*), de código abierto, ideado para programar en Java, pero con soporte para el resto de lenguajes de programación (Javascript, C, C++…). Lanzado por primera vez en Noviembre de 2001, y desarrollado por el Consorcio de Eclipse.org gracias a la inversión de 40 millones de dólares por parte de IBM en el proyecto, es uno de los entornos de programación más usados, posee una gran cantidad de herramientas, vistas, *plugins*, y otros contenidos que lo hacen ideal tanto para el programador iniciado como para el experto (Gallardo, 2012).

En el desarrollo del trabajo que nos ocupa, usaremos las vistas y *plugins* antes mencionados. Las primeras, para cambiar el entorno a uno adaptado al lenguaje Javascript, y los *plugins*, para conectar *Eclipse* con el repositorio de *Github*, y automatizar el proceso de programación.

C. CONFIGURACIÓN DE UN ENTORNO DE INTEGRACIÓN CONTINUA

1. Descripción del entorno a configurar

A continuación, se expondrá como instalar las herramientas analizadas anteriormente, en qué orden, y las configuraciones que hay que realizar en cada una de ellas para que el entorno de integración continua que se está intentando implementar funcione correctamente. El entorno quedará configurado de la siguiente forma:

-Un servidor en línea que se encargará del proceso de integración y despliegue continuo, en **Google Cloud Platform**, en el cual quedarán instalados tanto **Apache** y **Tomcat** (servidores web para el despliegue de la aplicación) como **Jenkins** (herramienta encargada de integrar y desplegar el código).

-Un repositorio en línea creado en **Github**, en el que se almacenarán las diferentes versiones de código que los programadores vayan creando.

-El IDE **Eclipse** instalado en cada uno de los ordenadores de los diferentes desarrolladores que participen en el proyecto.

2. Creación de servidor en Google Cloud Platform

Primero empezaremos creando y configurando el servidor que se encargará de la integración y despliegue continuos. Para ello, lo primero que se debe hacer es crearnos una cuenta en **Google Cloud** (Google, 2019a). Se solicitará al usuario un número de tarjeta para la facturación, pues el servicio es de pago, aunque no se cobrará nada pues, tal y como se explicó en el análisis de esta herramienta, *Google* ofrece un crédito de unos 260 € al año para gastar en esta herramienta. Una vez creada la cuenta, se mostrará una interfaz como la que se puede ver en la figura 9.

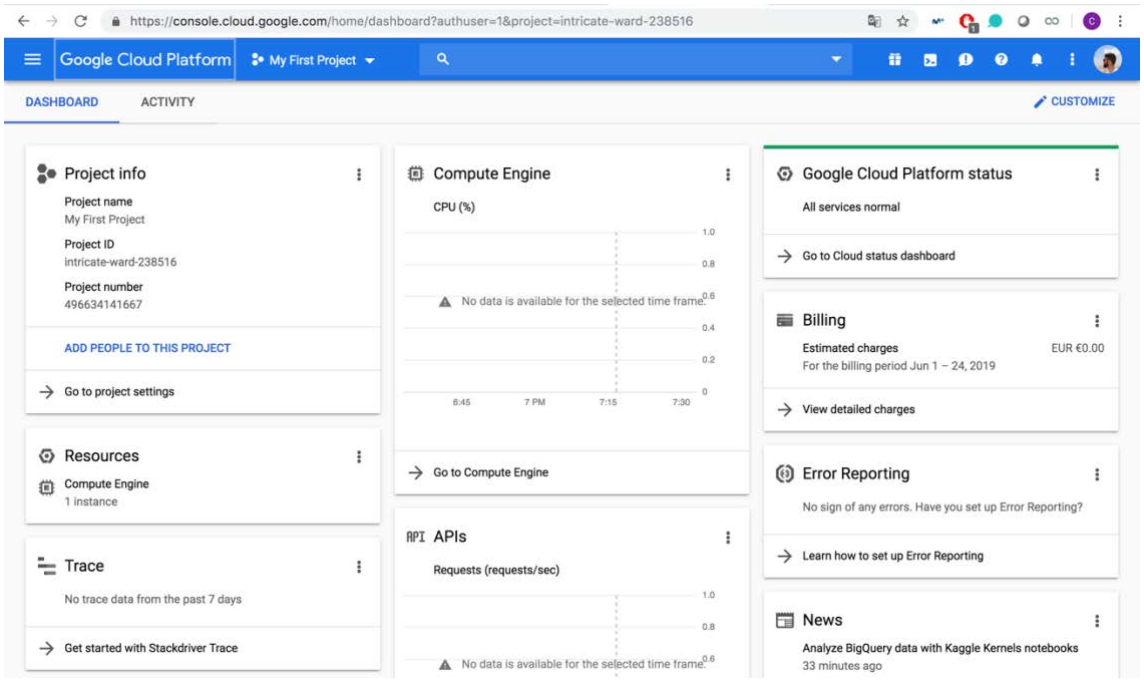


Figura 9. Interfaz de Google Cloud Platform (Elaboración propia)

Una vez dentro de esta interfaz, hay que buscar en el menú desplegable de la izquierda la aplicación ***Compute Engine***, y clicar en ***VM Instances***, donde se puede crear la instancia del servidor online que se requiere para la realización del proyecto (tal y como se muestra en la figura 10).

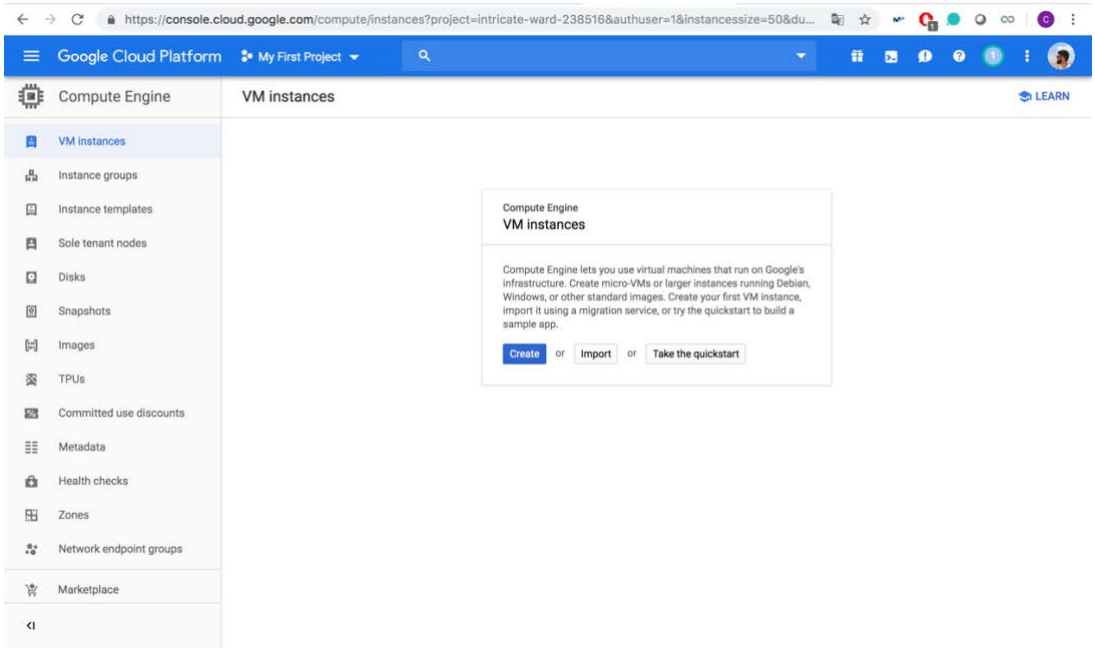


Figura 10. Creación de instancia en la aplicación Compute Engine (Elaboración propia)

Creada la instancia, se le pone un nombre, y se le asignará la siguiente configuración (resumida gráficamente en las figuras 11 y 12):

-*Región y zona*: Europe West (London).

-*Machine type*: 1 CPU (3,75GB de memoria).

-*Imagen de SO*: Para la realización de este trabajo se ha escogido **Ubuntu 16.04 LTS**, aunque *Google* provee al usuario de una gran cantidad de imágenes que elegir.

-*Firewall*: es importante marcar las dos casillas que se exponen, para permitir el tráfico *HTTP* y *HTTPS* dentro del servidor.

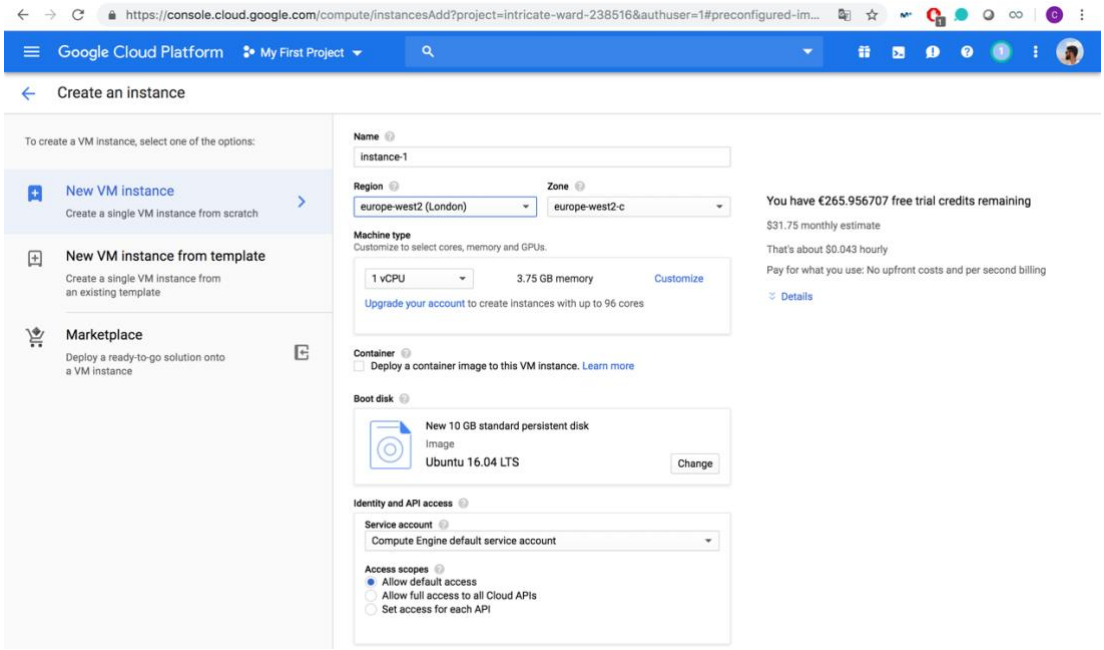


Figura 11. Configuración de la instancia en la aplicación Compute Engine (Elaboración propia)

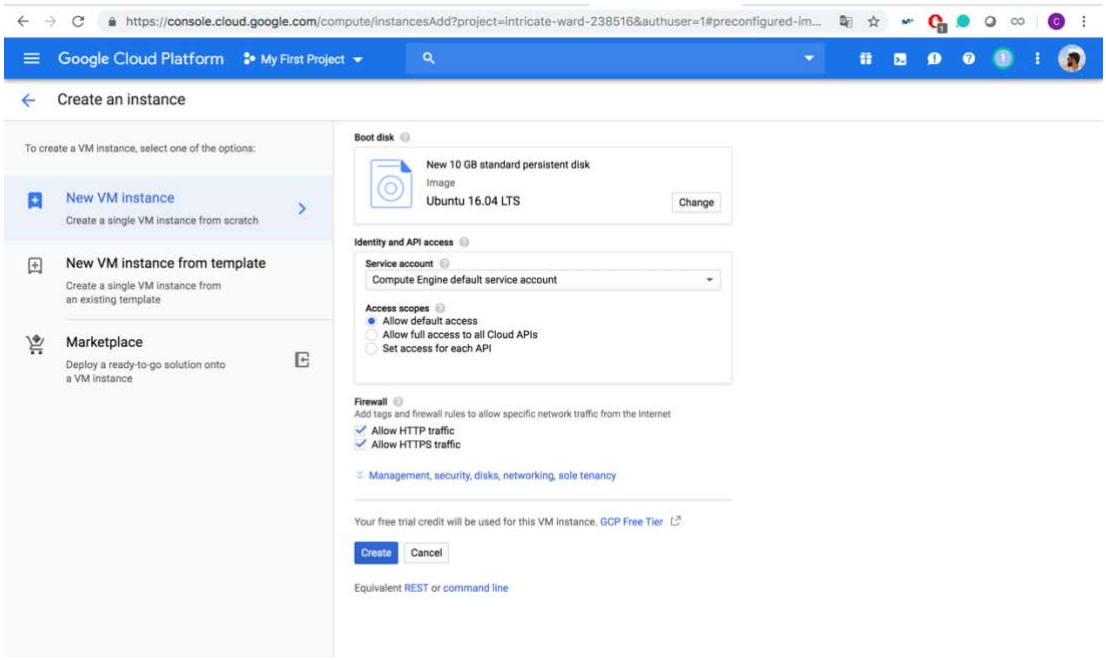


Figura 12. Configuración de la instancia en la aplicación Compute Engine (Elaboración propia)

Guardada la configuración y creada la instancia, se mostrará una lista con todas las instancias que hay creadas. Para correr la instancia que se creó anteriormente, se seleccionará su *checkbox* y se clicará en el botón de *play* en la barra de opciones que se muestra en la parte superior de la pantalla que se puede ver en la figura 13. Por último, para poder entrar al terminal del servidor para realizar cualquier configuración o instalación, una vez la instancia esté corriendo, se habilitará el botón de *SSH* de la instancia.

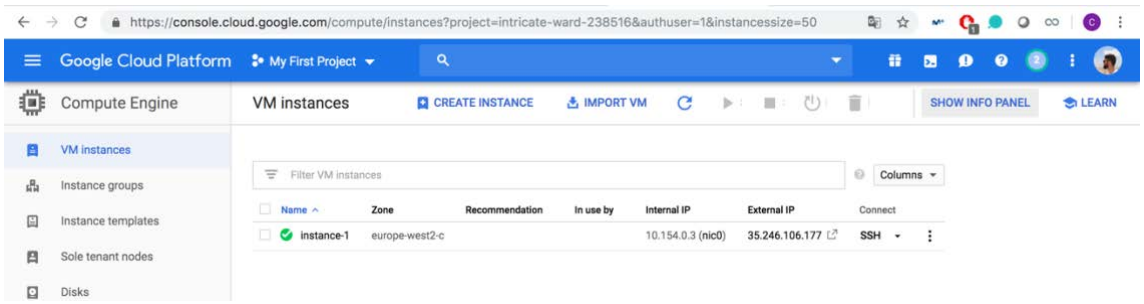


Figura 13. Instancia creada en la aplicación Compute Engine (Elaboración propia)

Al pulsar dicho botón, se abrirá otra ventana en el navegador, donde se mostrará el terminal de la instancia. Antes de realizar ninguna instalación o modificación en la

configuración del servidor, hay que teclear el siguiente comando para actualizar todos los paquetes que este sistema operativo lleva instalados por defecto (figura 14):

```
sudo apt-get update
```

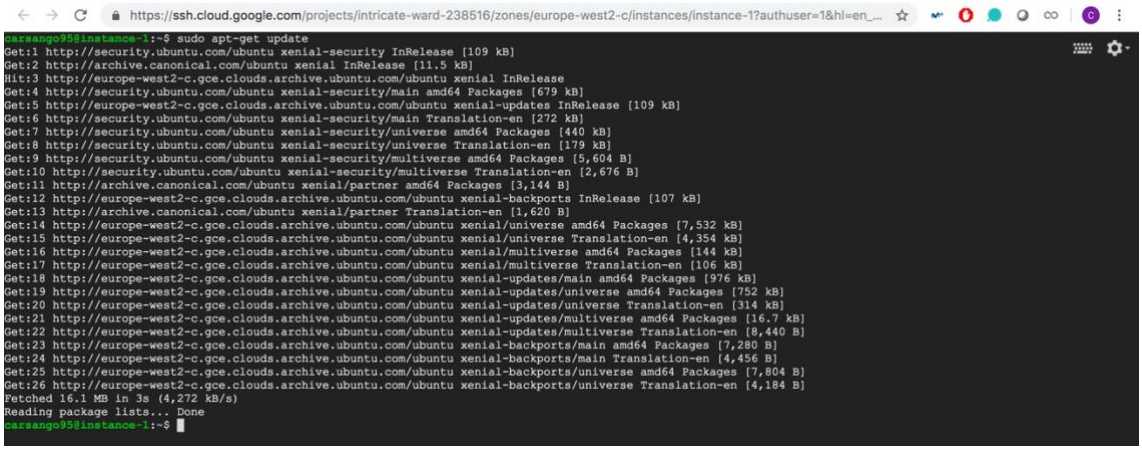


Figura 14. Terminal de la instancia creada en la aplicación Compute Engine (Elaboración propia)

3. Instalación de Java y Apache2

Una vez creado y configurado el servidor, se procederá a instalar las herramientas que ya se han mencionado anteriormente, empezando por el kit de desarrollo de Java (de forma abreviada ***JDK***, del inglés *Java Development Kit*), pues sin él, todas las herramientas que se instalarán que están basadas en Java no funcionarían correctamente.

Para ello, se debe teclear los comandos que se muestran a continuación:

```
sudo add-apt-repository ppa:openjdk-r/ppa
```

```
sudo apt update
```

```
sudo apt-get install openjdk-11-jdk
```

Con estos comandos, se consigue crear el repositorio local para el *JDK*, se instalan actualizaciones, y finalmente se instalará la versión más actual del *JDK* (en este caso, la versión 11). Tecleando el comando que se puede ver en la figura 15, se mostrará en la pantalla la versión que hay instalada en la instancia; como se puede apreciar, la versión instalada es la 11.0.4, con lo que el *JDK* se ha instalado correctamente

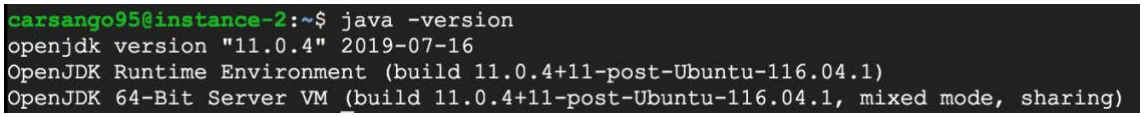


Figura 15. Terminal de la instancia creada en la aplicación Compute Engine (Elaboración propia)

A continuación, se procederá a instalar el servidor ***Apache***, de forma muy sencilla, tecleando los comandos que se exponen a continuación en el terminal del servidor:

```
sudo apt-get install apache2
```

```
sudo ufw allow "Apache"
```

```
sudo systemctl status apache2
```

Así, se consigue instalar *Apache* en el servidor, permitir el tráfico web a través de este, y comprobar que *Apache* está ejecutándose correctamente.

Por último, para comprobar su correcto funcionamiento en la web, se tecleará el siguiente comando para comprobar la IP de la instancia que hemos creado, y seguidamente se introducirá dicha IP en el navegador, donde, si todo ha ido correctamente, se mostrará la página por defecto de *Apache* (figura 16):

```
curl -4 icanhazip.com
```

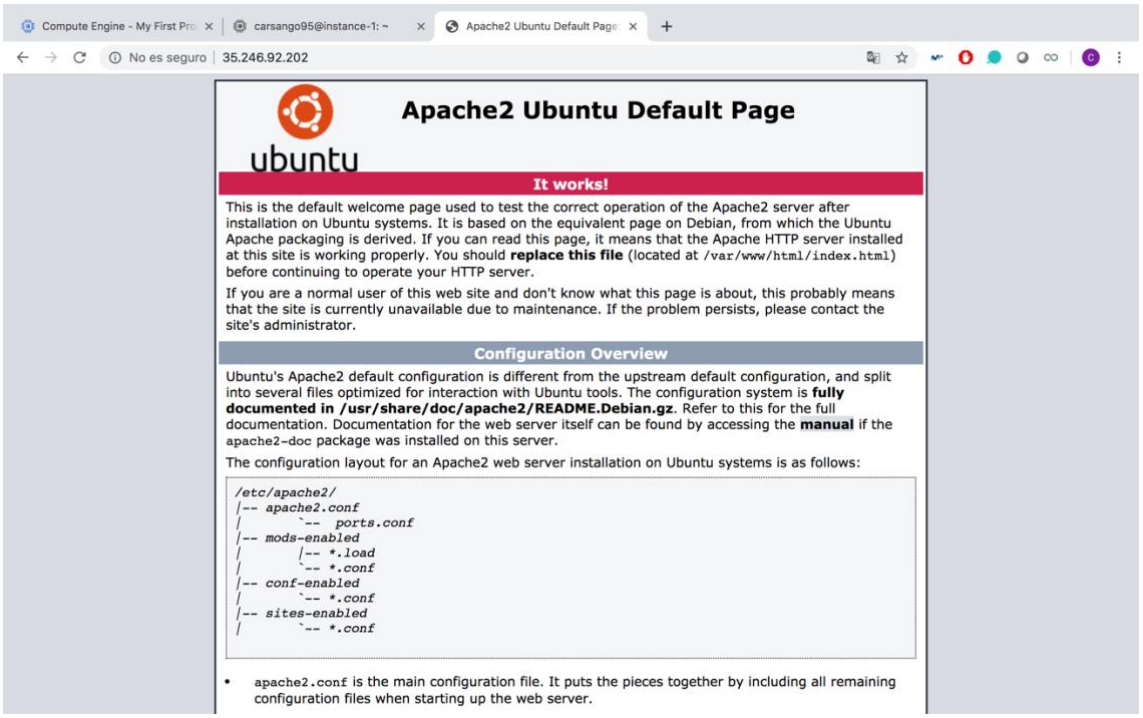


Figura 16. Página por defecto de Apache (Elaboración propia)

4. Instalación y configuración de Tomcat 9

La siguiente herramienta que instalaremos será ***Tomcat***, eligiendo la última versión de este. Antes de su instalación, crearemos en nuestro servidor un nuevo grupo y usuario de tomcat (que estará dentro de dicho grupo) en la carpeta donde posteriormente instalaremos la herramienta:

```
sudo groupadd tomcat
```

```
sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
```

Con esta configuración previa, ya se puede proceder a instalar *Tomcat*. Para ello, hay que acceder a la carpeta *tmp* con el comando que se puede ver a continuación (esta es una carpeta ideal para guardar archivos temporales, como el *.tar* que se descargará a continuación, que una vez descomprimido no será de ninguna utilidad):

```
cd /tmp
```

Seguidamente, hay que descargar la herramienta con el siguiente comando:

```
curl -O
http://mirror.cc.columbia.edu/pub/software/apache/tomcat/tomcat-9/v9.0.21/bin/apache-tomcat-9.0.21.tar.gz
```

En este caso, la ultima versión es la 9.0.21; antes de utilizar el comando, hay que comprobar, entrando en el siguiente link, que no existan versiones más recientes, pues si no el servidor no encontrará la ruta: http://mirror.cc.columbia.edu/pub/software/apache/tomcat/tomcat-9/

Si existiese una nueva versión, se debe cambiar el comando mencionado anteriormente, por la ruta del *.tar* de esta.

A continuación, se creará la carpeta *tomcat*, donde hay que instalar la herramienta, y más adelante se podrán encontrar todos los archivos de configuración relacionadas con esta, y se descomprimirá el archivo descargado anteriormente:

```
sudo mkdir /opt/tomcat
```

```
sudo tar xzvf apache-tomcat-9*tar.gz -C /opt/tomcat --strip-components=1
```

Una vez instalado *Tomcat*, empieza el proceso de configuración de este. Lo primero que se debe cambiar son los permisos del usuario sobre la herramienta, para poder realizar todas las funciones necesarias para el desarrollo del proyecto. Con los comandos que se muestran a continuación, se entrará en la carpeta de *tomcat* que se creó previamente, y se modificarán los permisos que el usuario tiene sobre *tomcat:*

```
cd /opt/tomcat
```

```
sudo chgrp -R tomcat /opt/tomcat
```

```
sudo chmod -R g+r conf
```

```
sudo chmod g+x conf
```

```
sudo chown -R tomcat webapps/ work/ temp/ logs/
```

En el siguiente paso, "se le indicará a *Tomcat* donde está instalado *Java"*. Para ello, se tecleará el comando que se puede ver en la figura 17, que devolverá la ruta de instalación de *Java*.

```
carsango95@instance-1:/opt/tomcat$ sudo update-java-alternatives -l
java-1.11.0-openjdk-amd64      1111       /usr/lib/jvm/java-1.11.0-openjdk-amd64
```

Figura 17. Terminal del servidor (Elaboración propia)

Con dicha ruta obtenida, se tecleará el siguiente comando para crear el archivo de configuración *tomcat.service*, donde hay que escribir todo lo que se muestra en la figura 18. En la variable *JAVA_HOME*, se pegará la ruta que se obtuvo en el paso anterior (en caso de que no sea la misma que la de la figura).

```
sudo nano /etc/systemd/system/tomcat.service
```

```
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking

Environment=JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
Environment=CATALINA_PID=/opt/tomcat/temp/tomcat.pid
Environment=CATALINA_HOME=/opt/tomcat
Environment=CATALINA_BASE=/opt/tomcat
Environment='CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC'
Environment='JAVA_OPTS=-Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom'

ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

User=tomcat
Group=tomcat
UMask=0007
RestartSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

Figura 18. Terminal del servidor (Elaboración propia)

Para que todos estos cambios surtan efecto, y comprobar que *Tomcat* sigue en correcto funcionamiento, se teclearán los siguientes comandos en la terminal:

```
sudo systemctl daemon-reload
```

```
sudo systemctl start tomcat
```

```
sudo systemctl status tomcat
```

También es necesario configurar el *firewall* del servidor para permitir el tráfico web en el puerto en el que actúa *Tomcat*, con el siguiente comando:

```
sudo ufw allow 8080
```

Con estos cambios realizados, se procederá a comprobar que *Tomcat* funciona desde el navegador. Para ello, hay que ingresar en la barra de búsqueda la IP del servidor, seguido de *:8080*. Si todo ha ido correctamente, debería mostrarse una página como la que se puede ver en la figura 19.

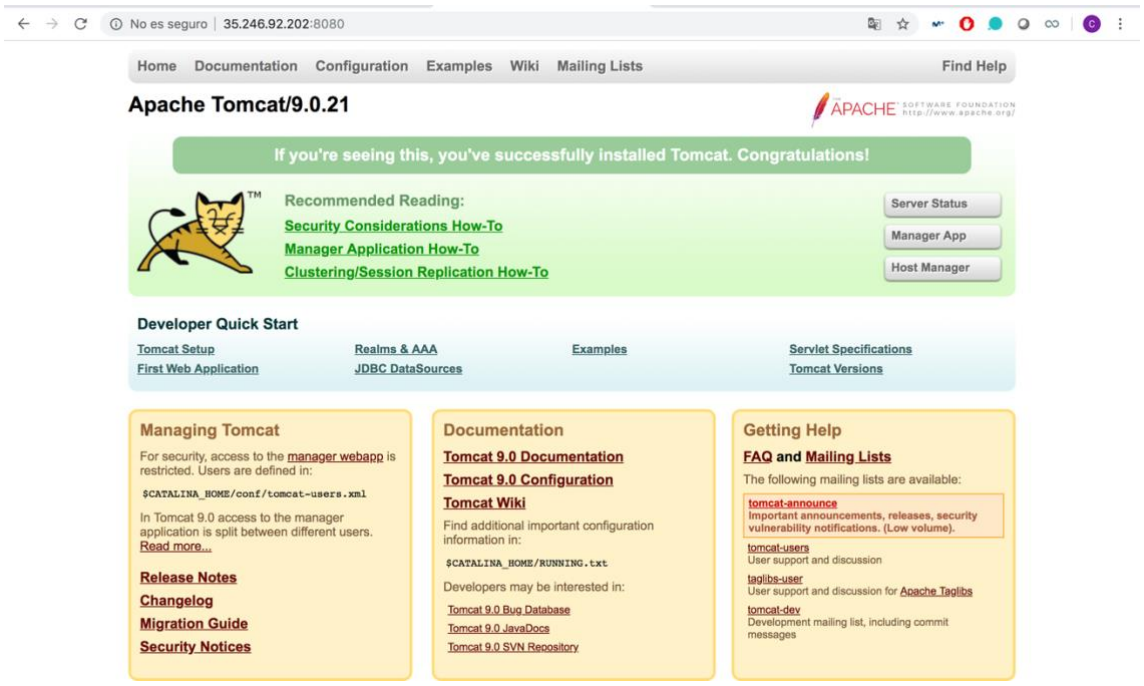


Figura 19. Página principal de Tomcat (Elaboración propia)

Una vez hemos comprobado que *Tomcat* funciona correctamente, se tecleará el siguiente comando para que este se ejecute cada vez que se inicie el servidor/instancia:

```
sudo systemctl enable tomcat
```

Aunque ya podamos ver la página principal de *Tomcat*, el usuario todavía no tiene permisos para actuar como *Manager* (rol que permitirá desplegar aplicaciones en el servidor web). Para ello, se debe modificar el archivo *tomcat-users.xml*, en el que se indicará un nombre de usuario, una contraseña, y los permisos que se quieren otorgar,

que serán, en este caso, *manager-gui* y *admin-gui*; se tecleará el siguiente comando en la terminal, y en el archivo que se nos abrirá, se escribirán, debajo de todos los comentarios que vienen por defecto en el archivo, dentro de las etiquetas de *tomcat-users*, las últimas líneas de código que se pueden apreciar en la figura 20, poniendo el nombre de usuario y contraseña que se deseen. Es importante anotarlas, pues serán las credenciales para poder acceder más tarde a la *Manager App* de *Tomcat*.

```
sudo nano /opt/tomcat/conf/tomcat-users.xml
```

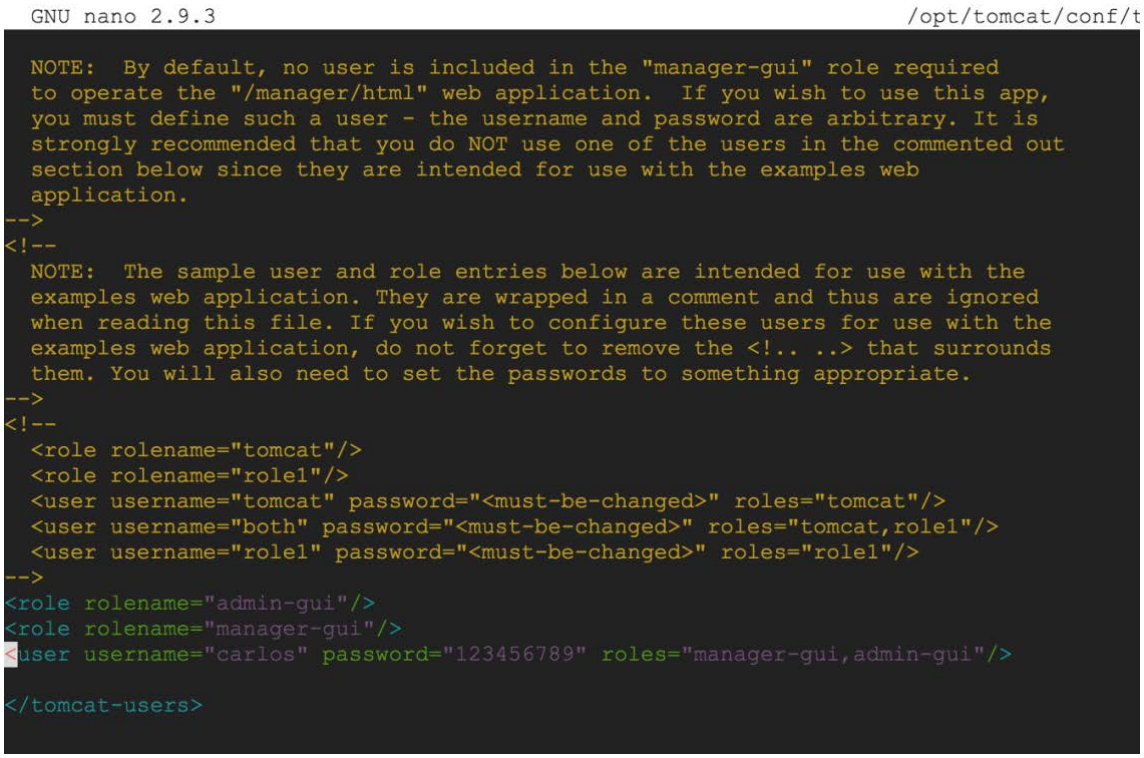


Figura 20. Ejemplo de configuración del archivo tomcat-users.xml (Elaboración propia)

Por defecto, *Tomcat* restringe el acceso al Manager a conexiones que vengan desde el propio servidor desde donde se está ejecutando este. Como en este proyecto se está trabajando con una conexión remota a este servidor, se debe modificar el archivo *context.xml* para quitar esta restricción. Para ello, hay que usar los siguientes comandos, que abrirán este archivo, tanto el de la carpeta *manager,* como el de la de *host-manager* (es necesario modificar ambos para que funcione).

```
sudo nano /opt/tomcat/webapps/manager/META-INF/context.xml
```

```
sudo nano /opt/tomcat/webapps/host-manager/META-INF/context.xml
```

Una vez dentro del archivo, lo único que hay que hacer es comentar la siguiente línea de código, poniendo las etiquetas de comentario antes y después de la sentencia que se puede ver a continuación (<!-- y -->), tal y como se puede ver en la figura 21.

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
```

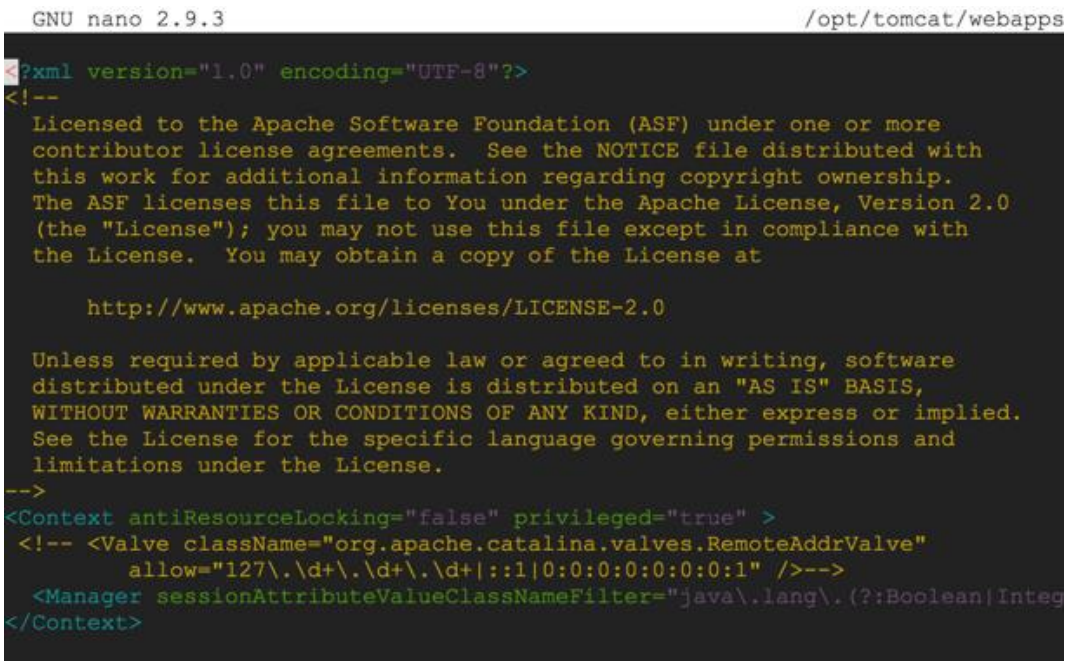


Figura 21. Ejemplo de configuración del archivo tomcat-users.xml (Elaboración propia)

Para terminar con la instalación de *Tomcat*, hay que reiniciarlo con el siguiente comando para que todos los cambios se efectúen:

```
Sudo systemctl restart tomcat
```

5. Creación de repositorio en Github

Para crear un repositorio en el que se puedan subir las versiones del código en *Github*, lo primero que hay que hacer es crear una cuenta en este. Para ello, se debe ingresar en la página principal (Github Inc., 2019) y clicar en *Sign up*. Se solicitará al nuevo usuario un nombre de usuario, una cuenta de correo electrónico, y una contraseña. A continuación, se ofrecerá al usuario obtener una cuenta gratuita (la que escogeremos en este proyecto) o una cuenta Premium.

En la siguiente página, hay que crear el repositorio. Habrá que darle un nombre, una descripción, se escogerá si se prefiere que sea público (cualquier usuario puede ver este repositorio, y modificarlo siempre que le se le den permisos) o privado. En la figura 22 se muestra un ejemplo de la creación de un repositorio.

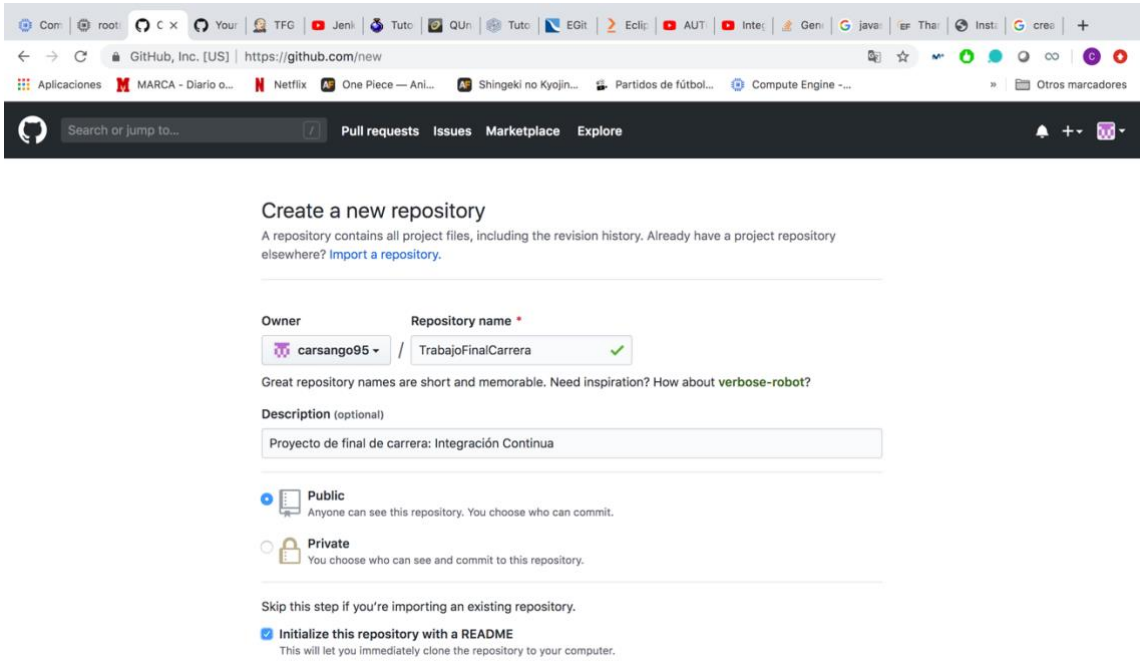


Figura 22. Creación de repositorio en Github (Elaboración propia)

Una vez creado el repositorio, se mostrará el *dashboard* del mismo, en el que ya se pueden ver los diferentes archivos que contiene el repositorio (de momento, solo se podrá ver el *README* que se genera por defecto). Ejemplo en la figura 23.

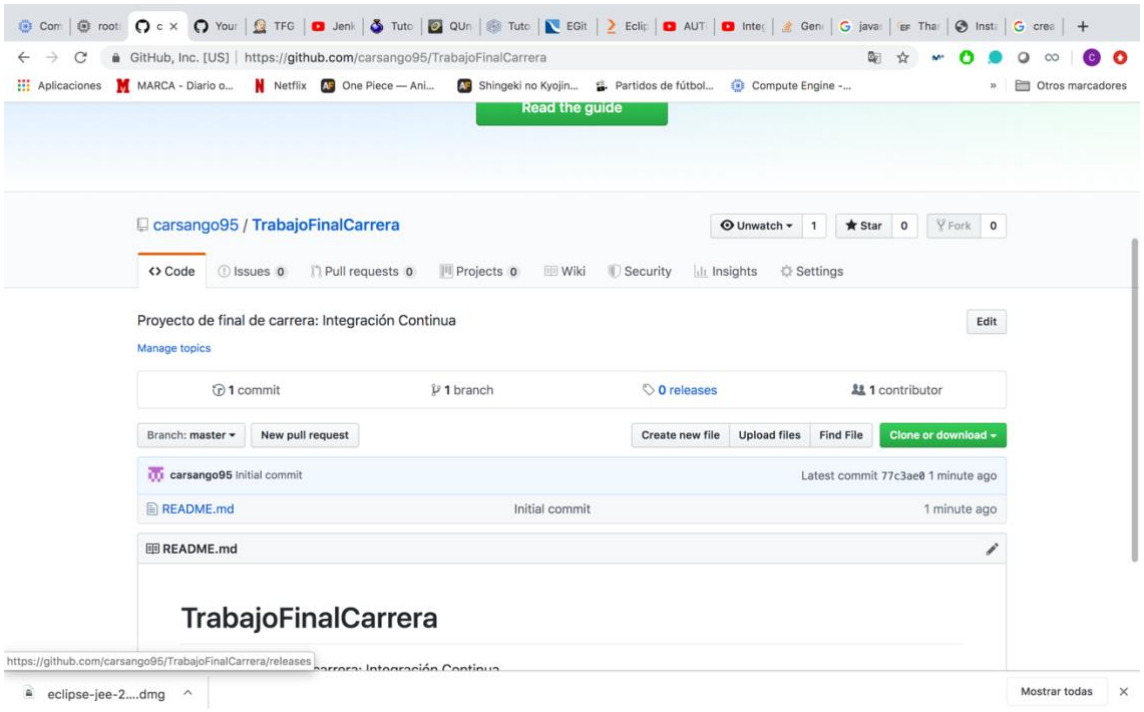


Figura 23. Dashboard de un repositorio en Github (Elaboración propia)

6. Instalación y configuración de Jenkins

Para instalar *Jenkins* en el servidor, lo primero será descargar el archivo *.war* que lo contiene, desde la página web oficial de la herramienta. Para ello, se utilizará el siguiente comando, que descargará la última versión de *Jenkins*:

```
wget http://updates.jenkins-ci.org/latest/jenkins.war
```

Lo siguiente que hay que hacer es copiar el archivo descargado en la carpeta *webapps* de *Tomcat*, lo que permitirá desplegar la aplicación desde este. Para ello, lo primero que habrá que hacer será utilizar el siguiente comando para actuar como el usuario *root* (si no, no se podrá acceder a esta carpeta):

```
sudo -s
```

Ya como *root*, se copiará el archivo en la carpeta *webapps*, se creará el directorio de *Jenkins* en la carpeta de *Tomcat*, se darán permisos al usuario para acceder a este directorio, y, por último, hay que reiniciar *Tomcat* para aplicar los cambios, todo esto introduciendo los siguientes comandos en el terminal:

```
cp jenkins.war /opt/tomcat/webapps/
cd /opt/tomcat
mkdir .jenkins
chown -R tomcat:tomcat /opt/tomcat/ .jenkins
sudo systemctl restart tomcat
```

A continuación, se debe comprobar que la ruta que hay escrita en el fichero *hudson.model.UpdateCenter.xml* es la correcta (este archivo ha dado problemas a la hora de instalar la aplicación). Usando el comando que se puede ver a continuación, se puede que comprobar que el contenido del fichero es el mismo que el que se puede ver en el ejemplo de la figura 24:

```
cd /opt/tomcat/.jenkins
nano hudson.model.UpdateCenter.xml
```

Figura 24. Ejemplo de configuración del archivo hudson.model.UpdateCenter.xml (Elaboración propia)

Antes de iniciar la aplicación, se debe usar el siguiente comando para averiguar la contraseña inicial que *Jenkins* tiene para el usuario administrador, ya que cuando se inicie, *Jenkins* la solicitará antes de poder realizar ningún cambio:

```
cd /opt/tomcat/.jenkins/secrets
cat initialAdminPassword
```

Se mostrará por pantalla dicha contraseña, se debe guardar para cuando sea requerida posteriormente. A continuación, se entrará en la herramienta *Jenkins* a través

del navegador, utilizando la siguiente ruta, poniendo la IP correspondiente a la maquina virtual creada (en el ejemplo se introduce la que se usa en este proyecto):

35.242.138.139:8080/jenkins

En la primera pantalla que aparece, se solicita la contraseña que se consiguió en el apartado anterior. Se debe ingresar dicha contraseña para continuar (figura 25).

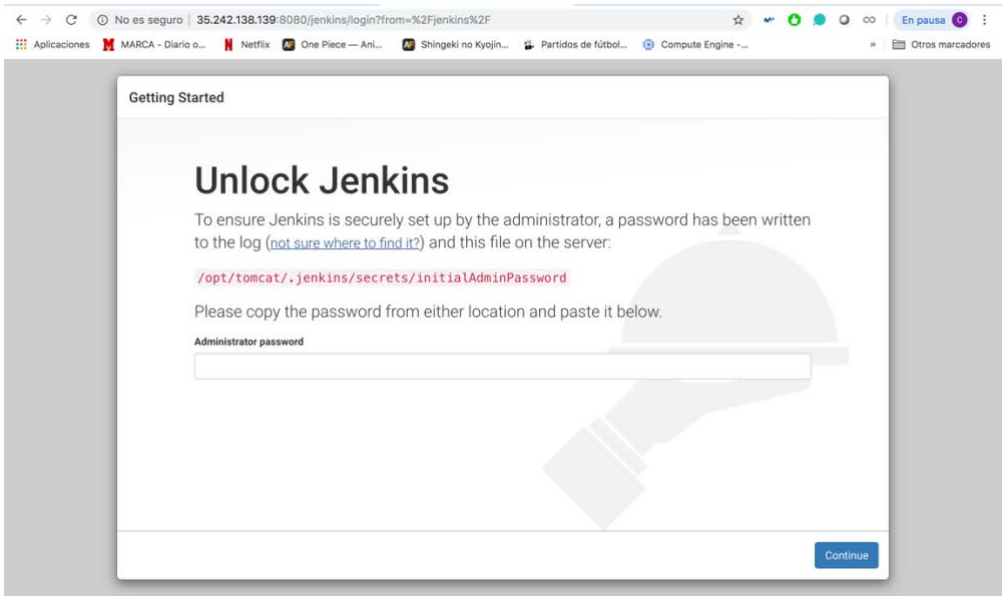


Figura 25. Solicitud de contraseña de Jenkins (Elaboración propia)

Una vez introducida la contraseña, se empezarán a cargar todos los ficheros de configuración de *Jenkins*, habrá que esperar a que termine. El siguiente paso será crear el usuario administrador. Es importante acordarse de las credenciales que se usen, pues cada vez que se quiera acceder a *Jenkins*, se van a solicitar (ejemplo de creación de usuario en la figura 26).

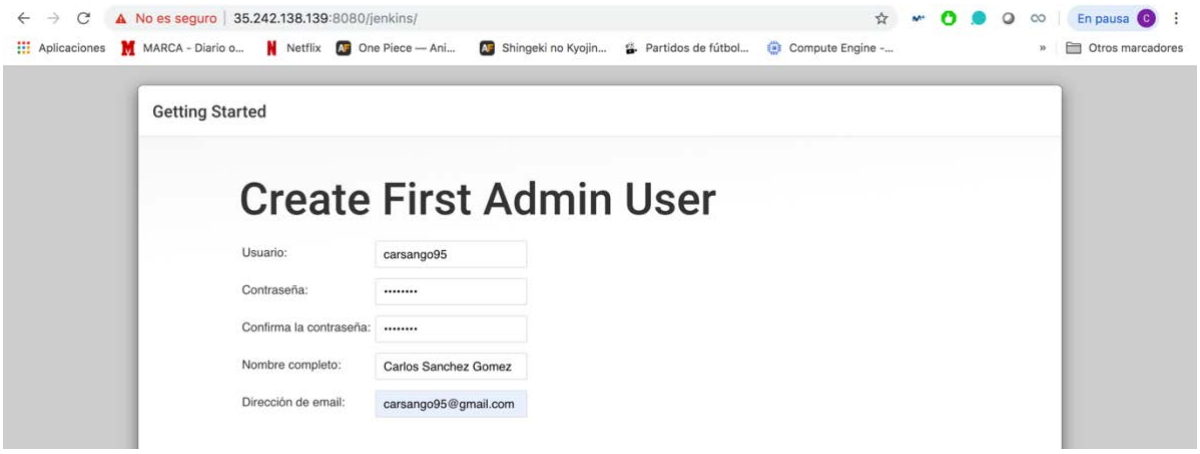


Figura 26. Creación de usuario administrador en Jenkins (Elaboración propia)

Terminado el proceso de creación de usuario, se mostrará la página de inicio de *Jenkins*, desde la que se pueden realizar multitud de tareas. De momento, solo se va a terminar de configurar *Jenkins* para que, más adelante, cuando empiece nuestro proyecto, funcione todo correctamente. Solamente faltaría instalar los *plugins* necesarios para conectar *Jenkins* con *Github*, y para poder desplegar aplicaciones web en *Tomcat*. Para ello, habrá que ir al menú de opciones de *Jenkins*, y acceder a *Administrar Jenkins* (ver en figura 27).

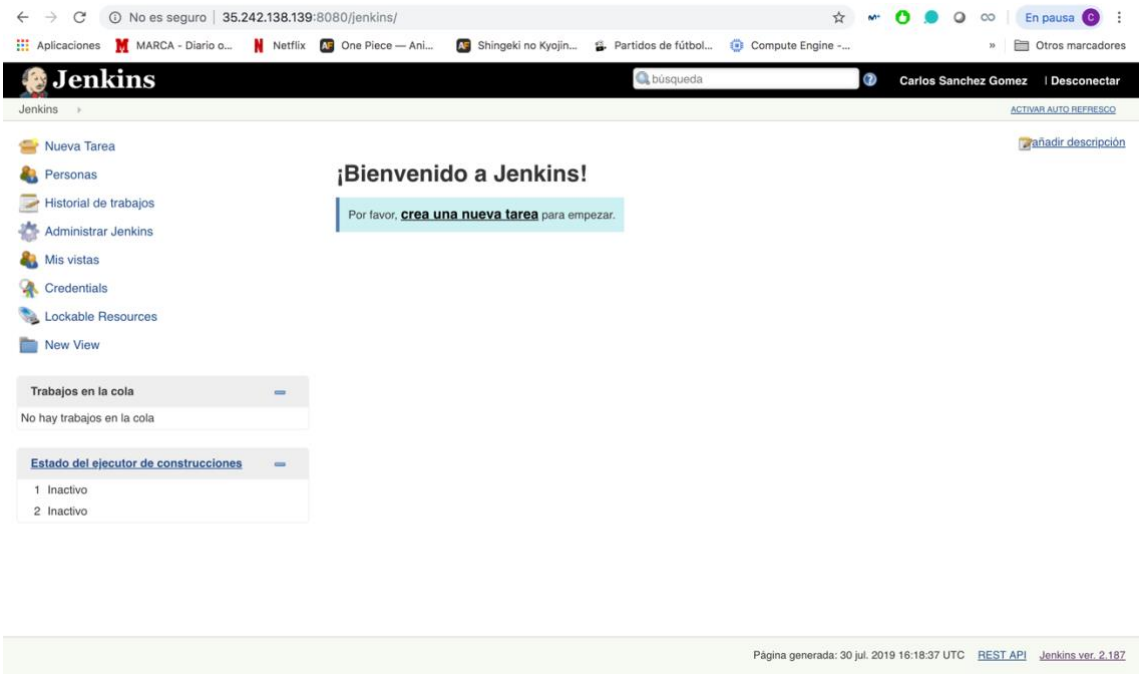


Figura 27. Pantalla de inicio de Jenkins (Elaboración propia)

Aquí se pueden ver todos los *plugins* que están instalados, y los que podrían ser instalados. Se utilizará el buscador para instalar los *plugins* que, como hemos comentado antes, permitan integrar *Jenkins* con *Github* y *Tomcat*. Se pueden ver en las figuras 28 y 29.

Figura 28. Plugins para la integración con Github (Elaboración propia)

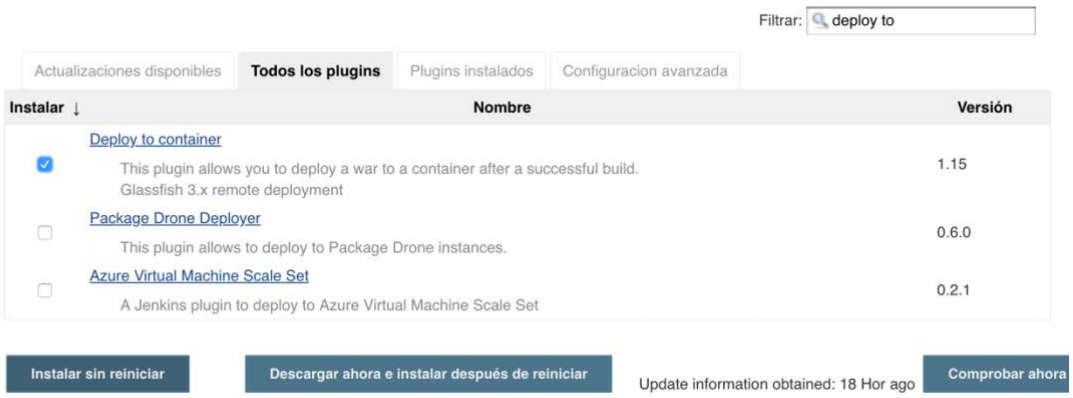


Figura 29. Plugins para la integración con Tomcat (Elaboración propia)

7. Instalación y configuración de Eclipse

7.1 Descarga

La última herramienta que habrá que descargar para el entorno de integración continua será *Eclipse*, la cual no será instalada en el servidor en línea, sino en local, puesto que será utilizada por los programadores para implementar el código del proyecto. Para descargar el programa, hay que ir a la página oficial de *Eclipse*, a la zona de descargas (Eclipse Foundation Inc., 2019).

En esta página se pueden encontrar diferentes versiones de *Eclipse*, cada una para un tipo de desarrollo distinto. Para el desarrollo de nuestro proyecto, se descargará

Eclipse IDE for Enterprise Java Developers, que contiene las funcionalidades necesarias para desarrollar una aplicación web, su despliegue, e integración con *Github*.

Una vez descargado el paquete, se instalará y seleccionará el *Workspace*, carpeta donde se guardarán, en local, los proyectos que se vayan creando en *Eclipse*. Realizados estos pasos, se abrirá *Eclipse* y, antes de realizar ninguna modificación, habrá la perspectiva a la de *Web*, tal y como se puede ver en las figuras 30 y 31.

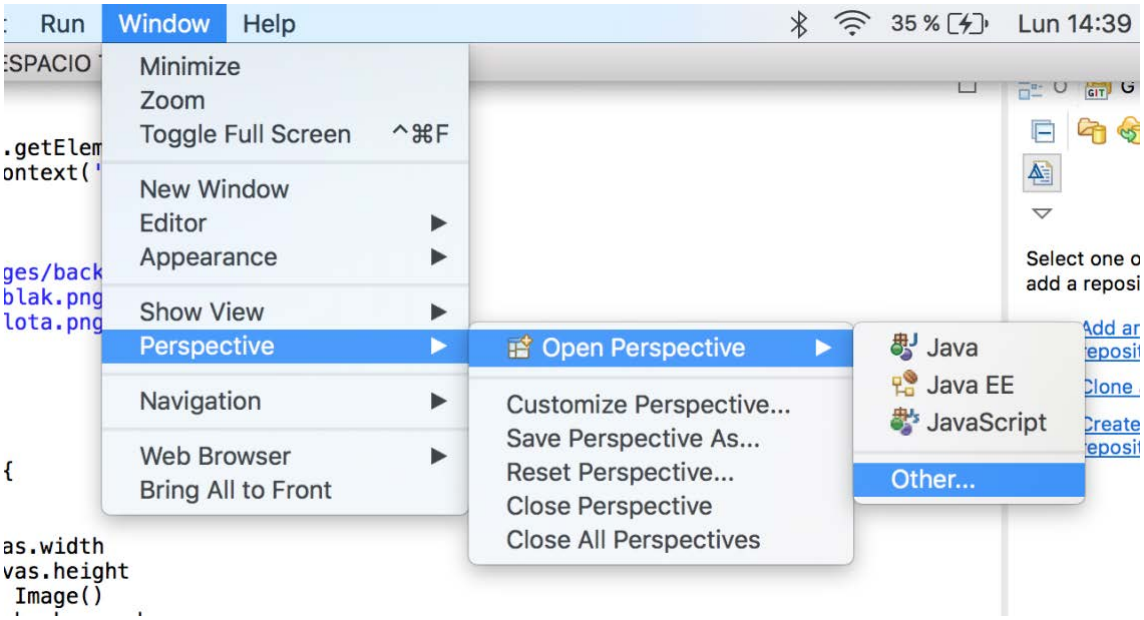


Figura 30. Ejemplo de cambio de perspectiva en Eclipse (Elaboración propia)

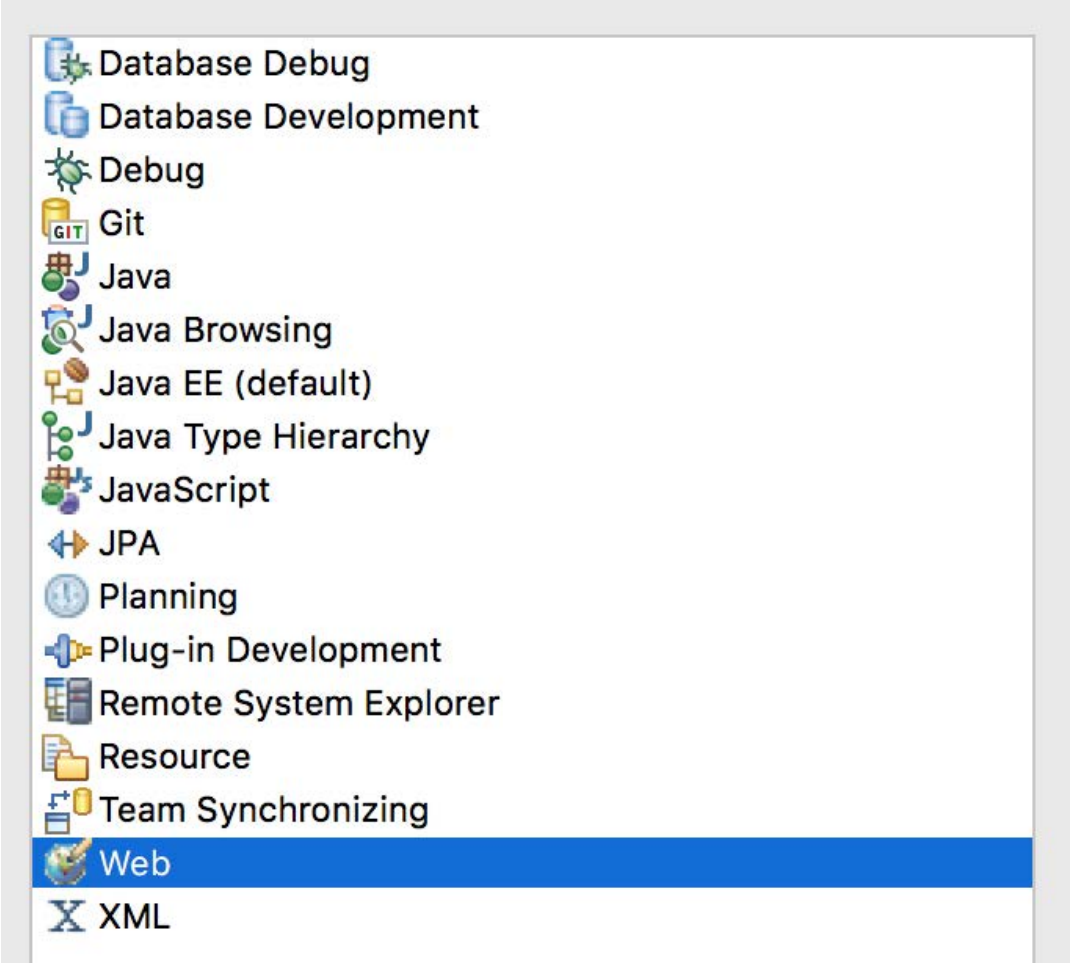


Figura 31. Perspectivas disponibles dentro de Eclipse (Elaboración propia)

7.2 Configuración de EGit

Después de instalar *Eclipse*, se continuará instalando el *plugin* de este que nos permite integrarlo con *Github*, para así poder sincronizar el progreso que se haga en *Eclipse*, con el repositorio que ya se había creado. Para ello, se debe clicar en la barra de opciones de *Eclipse*, en *Help*, y seguidamente en *Eclipse Marketplace*, donde se pueden ver y descargar todos los *plugins* disponibles para *Eclipse*. En el buscador, se tecleará *EGit*, para instalar la última versión del *plugin*. Una vez instalado, en la barra de opciones, habrá que clicar en *Window* → *Show View* → *Other*. En la ventana que se abre, hay que seleccionar la opción que vemos marcada en la figura 32.

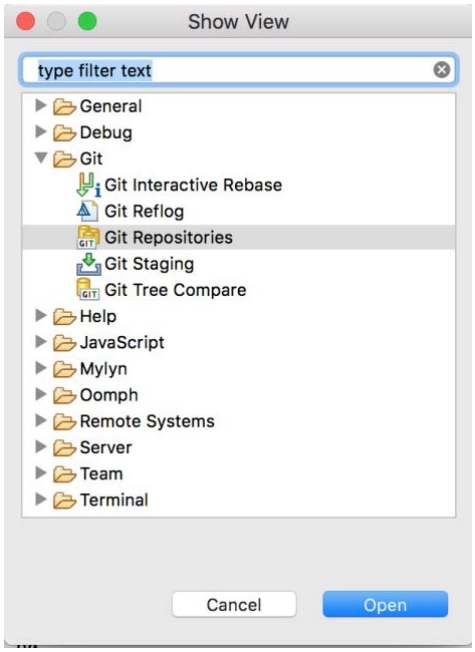


Figura 32. Vista de Git en Eclipse (Elaboración propia)

7.3 Clonado de repositorio

Como último paso de configuración del entorno, hay que clonar el repositorio que se creó en *Github*, para trabajar sobre él en *Eclipse*. Para ello, en la vista que se abrió en el paso anterior, hay que clicar en *Clone a Git repository*, y se mostrará una ventana como la de la figura 33, donde hay que rellenar los campos que se solicitan.

-*URL*: se copia la URL del repositorio en el navegador

-*Host*: github.com

-*Repository path*: como en el ejemplo, pero con el nombre de usuario y el nombre del repositorio que se haya elegido

-*Protocol*: https

-*User and password:* credenciales en *Github*

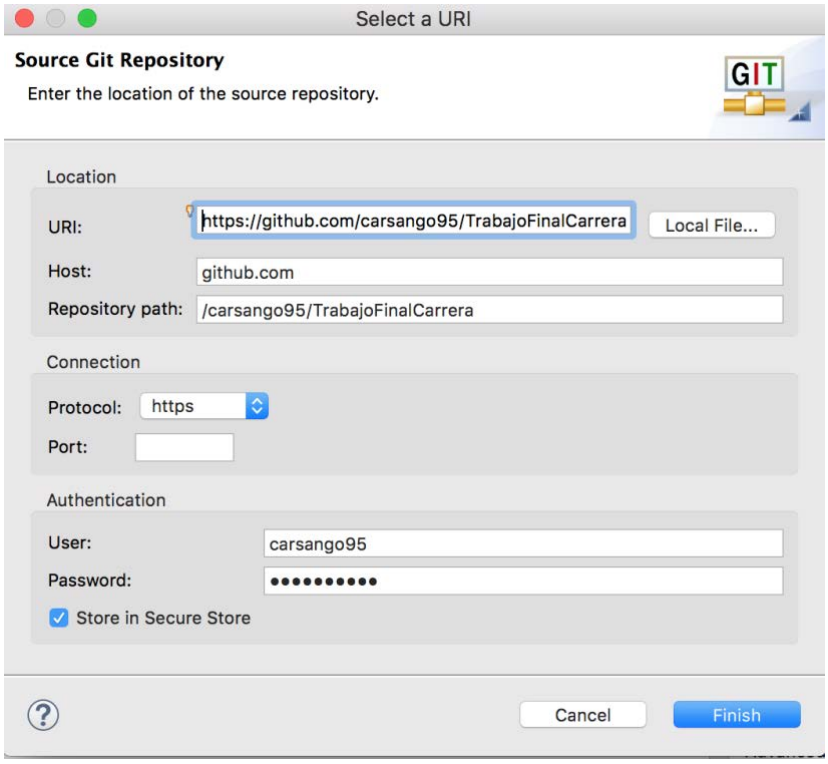


Figura 33. Clonado de repositorio (Elaboración propia)

Por ultimo, una vez clonado el repositorio, hay que clicar con el botón derecho en el repositorio que ha aparecido en la vista de *Git* en *Eclipse*, y seleccionar la opción de *Import Projects*, que importará en *Eclipse* los proyectos que contenga el repositorio de *Github* que se clonó anteriormente. Así, el desarrollador ya puede empezar a implementar su código en el proyecto que se acaba de importar en *Eclipse.*

D. CASO PRÁCTICO: UTILIZACIÓN DE UN ENTORNO DE INTEGRACIÓN CONTINUA PARA IMPLEMENTAR UN CÓDIGO JAVASCRIPT

1. Descripción del caso práctico: minijuego en Javascript

Una vez instalado y configurado el entorno de integración continua, ya se puede proceder a desarrollar el proyecto. El ejemplo que se llevará a cabo será el desarrollo de un sencillo juego programado en Javascript. El juego está compuesto básicamente por tres objetos: portero, pelota y *background*. El usuario manejará al portero con las flechas de dirección (izquierda y derecha), e intentará coger las pelotas que irán saliendo

de la parte inferior de la pantalla hacia la portería. Si el usuario pierde tres pelotas, el juego termina, obteniendo el usuario una puntuación igual al número de pelotas que ha conseguido salvar.

Para el desarrollo de este proyecto, se emulará la metodología *Agile*, en la que, como ya se ha explicado, se realizan diversas entregas de código (versiones/sprints), por lo que, para este ejemplo, se ha supuesto un escenario en el que el desarrollador realiza hasta tres entregas de código. Estas tres entregas o versiones van desde una funcionalidad básica de la aplicación, hasta la funcionalidad final que hemos explicado en el párrafo anterior. Se desglosarían de la siguiente forma:

-*Versión 1*: Aparecen todos los objetos del juego, los balones salen de cualquier zona de la parte inferior de la pantalla, si el portero no coge una pelota, termina la partida, no hay puntuación.

-*Versión 2*: Los balones solo van hacia la portería, a velocidad más lenta para que el usuario sea capaz de mantener una partida más larga, no hay puntuación.

-*Versión 3*: Si el usuario no coge un balón, tendrá dos oportunidades más. Cuando termina la partida, se muestra una puntuación que equivale al número de balones rescatados.

A continuación, se explicará, paso a paso, como hacer uso del entorno que acabamos de instalar, para desarrollar el proyecto que se ha descrito.

2. Creación de proyecto Jenkins

Antes de comenzar a implementar el código, hay que configurar *Jenkins* para que se sincronice con el repositorio de *Github*, y sea capaz de detectar cualquier cambio que se produzca en este. Estando en la pantalla de inicio de *Jenkins* (figura 34), hay que clicar en *Nueva Tarea* → *Crear un proyecto de estilo libre*.

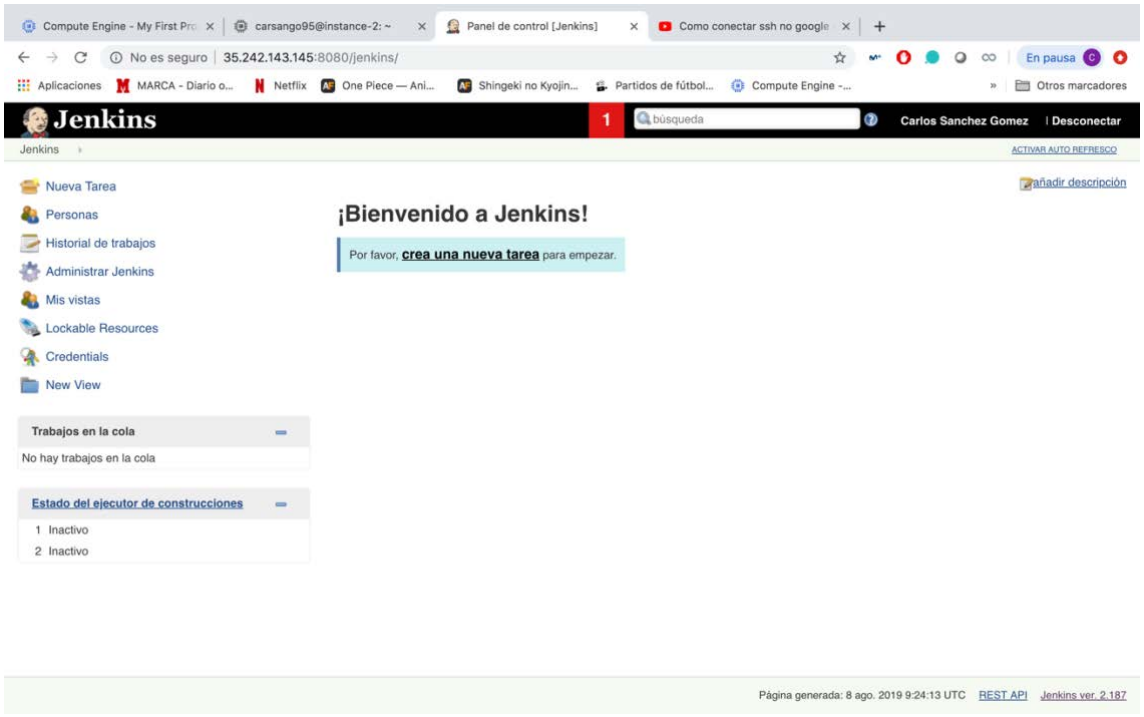


Figura 34. Pantalla de inicio de Jenkins (Elaboración propia)

Se mostrará a continuación la pantalla de configuración de proyecto de *Jenkins* (si fuese necesario realizar alguna modificación de dicha configuración una vez avanzado el proyecto, se puede acceder posteriormente). En esta pantalla de configuración hay que darle un nombre al proyecto, una descripción, pero lo más importante es que será aquí donde se le dirá a *Jenkins* que tiene que coger el proyecto de un repositorio de *Github*, y la ruta a dicho repositorio. Además de esto, también se configurará *Jenkins* para que, automáticamente, revise si se ha realizado alguna modificación en el repositorio, y compilar el código en busca de errores (el despliegue automático lo se configurará más adelante). En la figura 35 se puede ver un ejemplo de cómo configurar la sincronización con *Github* (cambiando siempre el nombre y ruta del repositorio por el nuestro), y en la figura 36 se configura *Jenkins* para que, cada 15 minutos, compruebe si se han realizado cambios en el repositorio.

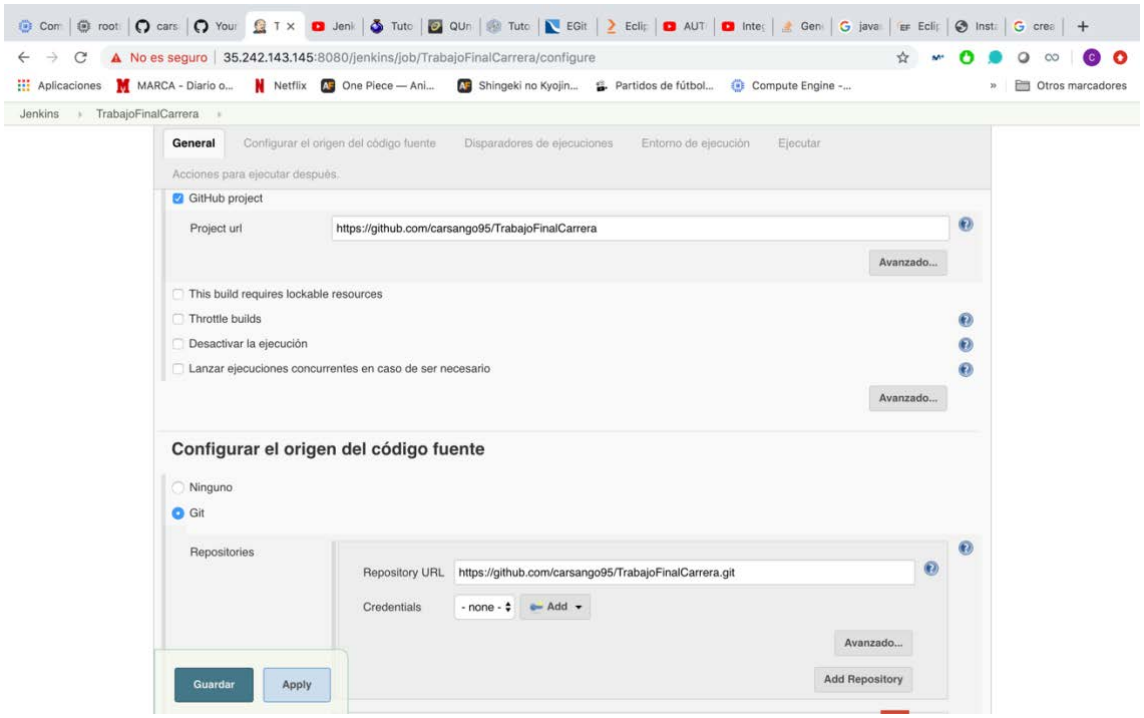


Figura 35. Pantalla de configuración de proyecto en Jenkins (Elaboración propia)

Figura 36. Pantalla de configuración de proyecto en Jenkins (Elaboración propia)

Una vez guardados los cambios que se han realizado en la configuración del proyecto, se creará el proyecto y su espacio de trabajo, dentro del cual se pueden encontrar los archivos que *Jenkins* ha rescatado del repositorio de *Github* que se ha indicado en la configuración.

Otra modificación previa que se debe hacer antes de empezar a implementar el proyecto es convertir el mismo a un proyecto de tipo *Maven* en *Eclipse*. Esto es necesario para poder exportar el proyecto como un *.war*, imprescindible para realizar

despliegues de aplicaciones web en *Tomcat*. Como se puede ver en la figura 37, es tan sencillo como clicar con el botón derecho en el proyecto en *Eclipse*, y en el apartado de *Configure*, seleccionar *Convert to Maven Proyect*. En la ventana que se abrirá después, basta con clicar en *Finish*.

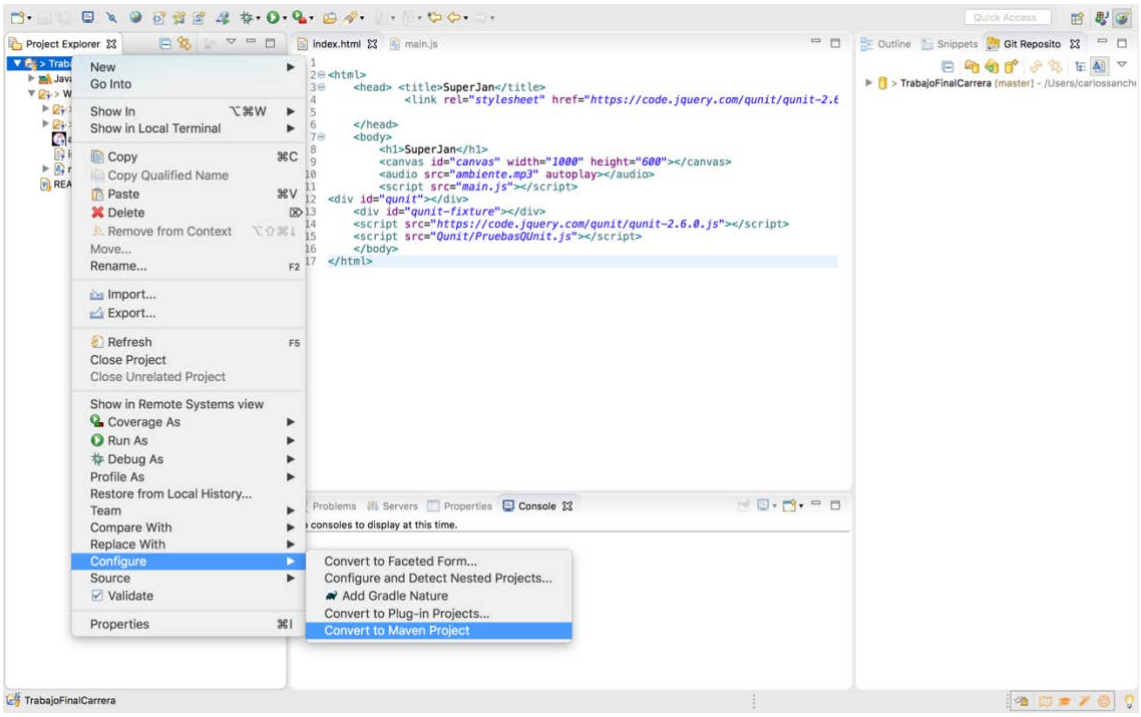


Figura 37. Convertir un proyecto en Maven en Eclipse (Elaboración propia)

3. Primera implementación de código e integración con el repositorio

Realizadas todas las configuraciones previas, se comenzará con el primer *sprint* o versión. El primer paso, que corre por la cuenta del desarrollador, consiste en implementar el código en *Eclipse*. Dentro de dicha implementación el desarrollador debe incluir también las pruebas de código. Como se ha explicado anteriormente, el *framework* que se utilizará para realizar las pruebas de código será *QUnit*, uno de los pocos *frameworks* de pruebas de código en *Javascript* con soporte para el usuario. A pesar de tener la ventaja de que su aprendizaje y uso se tornan bastante sencillos (además de que en la página oficial de *QUnit* se pueden encontrar tutoriales para aprender a realizar estas pruebas, en internet se pueden encontrar también multitud de páginas que también lo explican), tiene el inconveniente de no poder automatizar estas pruebas, al no existir en *Jenkins* el *plugin* correspondiente para lanzar estas pruebas cada vez que detecte un cambio en el repositorio de *Github*.

Si el proyecto que se está desarrollando fuera realizado en *Java*, se podrían realizar las pruebas en *JUnit*, un *framework* bastante conocido para realizar pruebas en dicho lenguaje, y del que si existe un *plugin* en *Jenkins* para que, automáticamente, se lancen estas pruebas cada vez que se realice una compilación de código, y se muestre en el proyecto en *Jenkins* un reporte con el número de pruebas correctas y fallidas.

A consecuencia de esto, todos los tutoriales que se pueden encontrar en la web sobre la utilización de *Jenkins* para realizar un proceso de integración continua, utilizan como ejemplo un código en *Java*, por lo que, ante la falta de tutoriales o ejemplos para realizar este proceso en un proyecto de *Javascript*, en este trabajo se expondrá una propuesta de cómo realizar un proyecto de *Javascript* de la forma más automatizada posible, y que pueda servir como guía o ejemplo a todo aquel usuario que se quiera lanzar a desarrollar en *Javascript*, y seguir el modelo de la metodología *Agile*.

Ante la incapacidad de *Jenkins* de, al menos de momento, realizar pruebas automatizadas para *Javascript* sobre *QUnit*, la metodología que llevaremos a cabo será la siguiente: el desarrollador, una vez implementado su código con sus correspondientes pruebas (como vimos en el análisis de *QUnit*, dichas pruebas se incluyen en el *index.html* del código *Javascript*), lo ejecutará en *Eclipse*, para comprobar el resultado de dichas pruebas. Si ninguna de dichas pruebas resulta fallida, procederemos a realizar el *commit* sobre el repositorio de *Github*. Se puede ver un ejemplo de la ejecución de dichas pruebas en la figura 38.

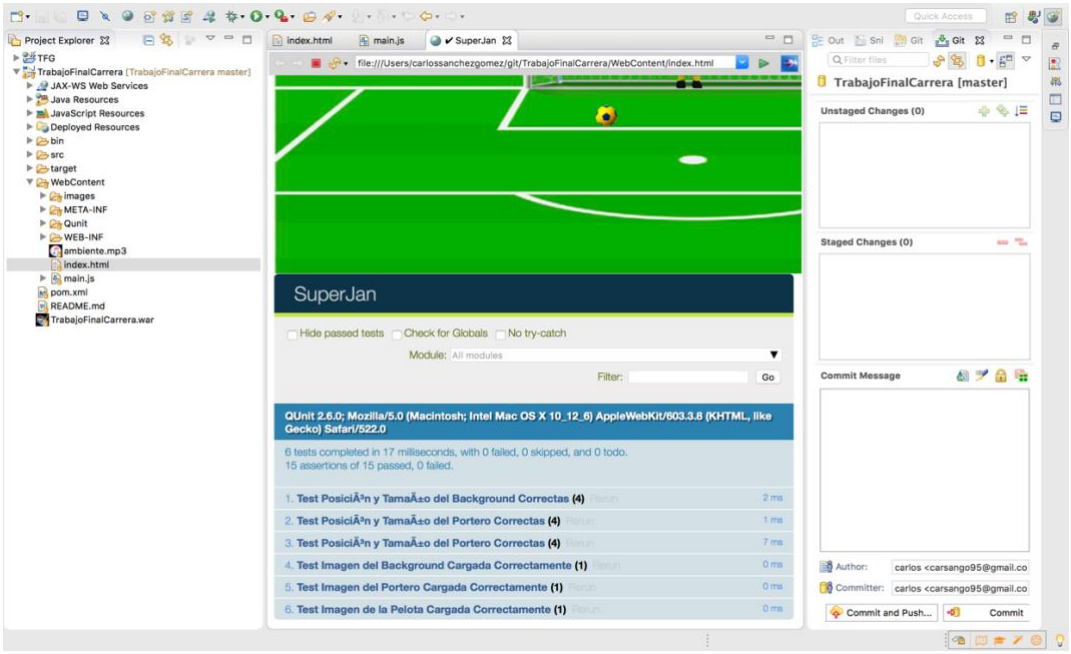


Figura 38. Ejecución de pruebas de QUnit en Eclipse (Elaboración propia)

Una vez implementado el código y ejecutadas las pruebas, se procederá a realizar el *commit* sobre el repositorio. Antes de ello, hay que configurar el proyecto para poder ser exportado como *.war*. Además de la conversión a proyecto *Maven* realizada en el apartado anterior, hay que clicar con el botón derecho en nuestro proyecto en *Eclipse*, y acceder a *Properties*. Aquí, como se puede ver en la figura 39, se debe acceder a *Maven* → *Project Facets*, y marcar la casilla de *Dynamic Web Module.*

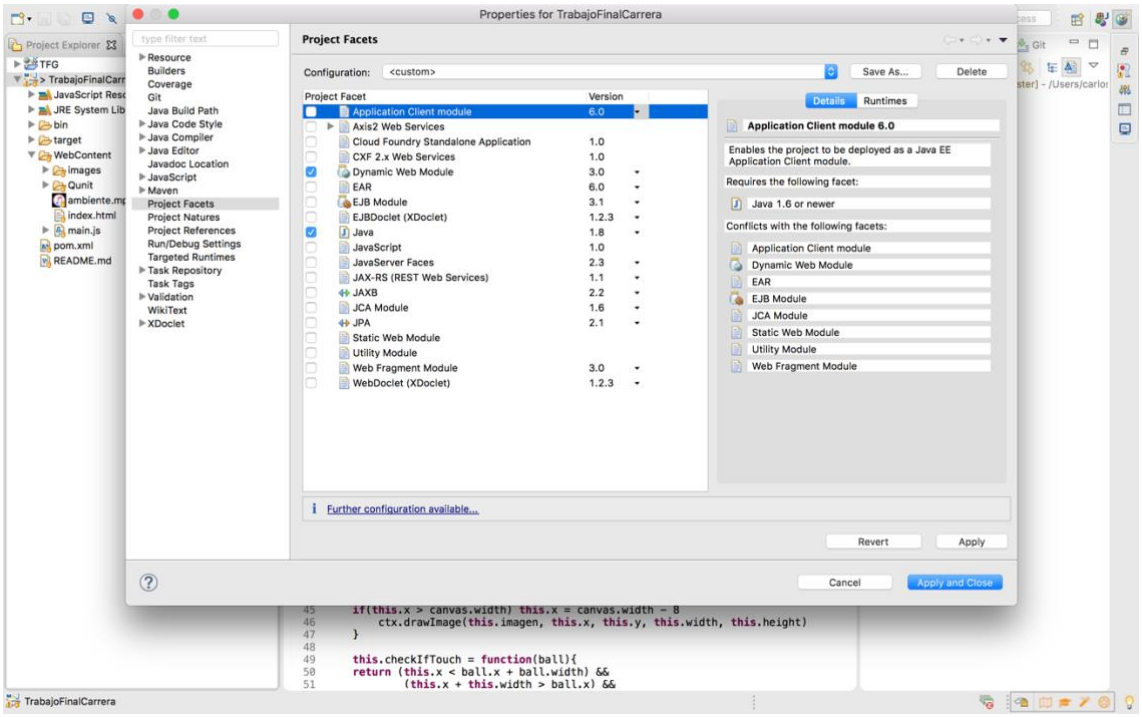


Figura 39. Configuración de proyecto como Dynamic Web Module (Elaboración propia)

Convertido a *Maven* y configurado como *Dynamic Web Module,* el proyecto ya está listo para ser exportado como *.war*. Para ello, se clicará con el botón derecho en el proyecto en *Eclipse*, seguidamente en *Export* → *WAR file* (ejemplo en figura 40). En la ventana que se abrirá, se elegirá una ubicación para el nuevo archivo.

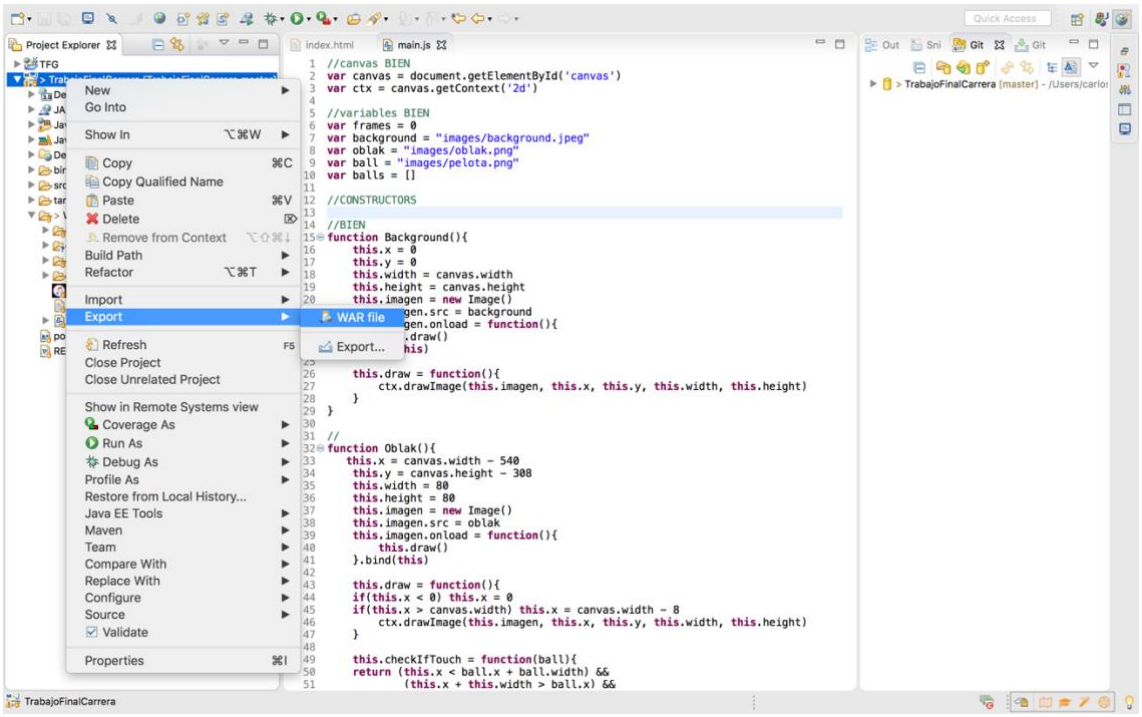


Figura 40. Exportar proyecto como .war (Elaboración propia)

Una vez creado el archivo, hay que arrastrarlo desde la carpeta del ordenador donde esté ubicado hacia *Eclipse,* en la carpeta del proyecto que se está desarrollando, de forma que quede como en el ejemplo de la figura 41, en el que podemos ver el archivo *TrabajoFinalCarrera.war* en la parte inferior de la carpeta.

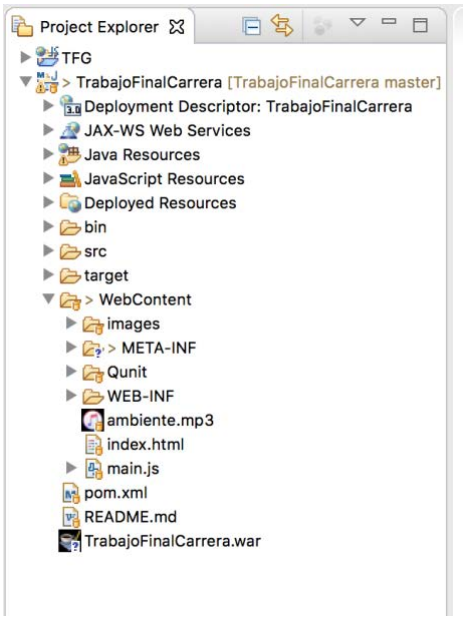


Figura 41. Archivo .war en la carpeta del proyecto en Eclipse (Elaboración propia)

Para realizar el *commit* en el repositorio, habrá que situarse en la ventana de trabajo de *Git* que se abrió anteriormente en *Eclipse*. El proceso a seguir es el siguiente:

-Todos los cambios que se hayan realizado en el proyecto que no estén en sincronía con el repositorio, aparecerán en la ventana de *Unstaged Changes*

-Para introducir dichos cambios en el repositorio, hay que arrastrar estos archivos desde la ventana *Unstaged Changes* hasta la de *Staged Changes*, de forma que quede como en el ejemplo de la figura 42.

-En la ventana de *Commit Message* se puede escribir un mensaje que quedará reflejado en *Github* en cada uno de los archivos que actualicemos en este *commit*.

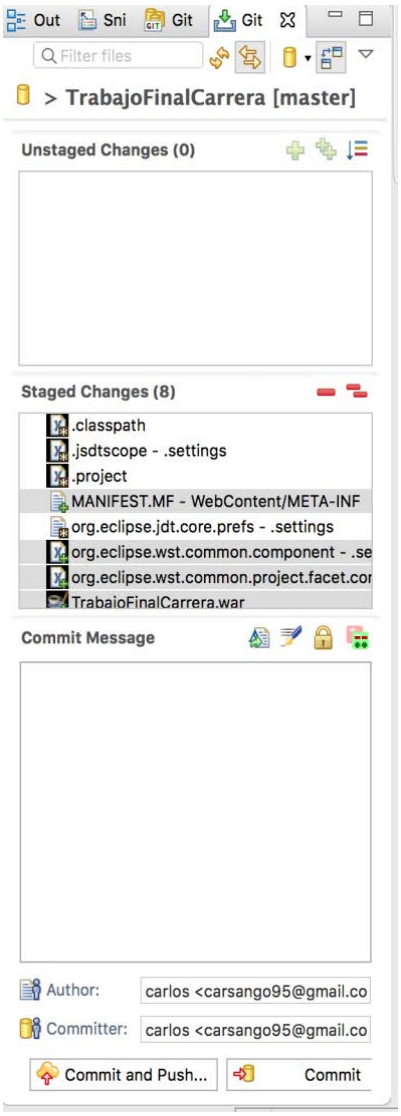


Figura 42. Ejemplo de commit dentro de Eclipse (Elaboración propia)

Realizado el *commit*, se actualiza el repositorio de *Github* con cada uno de los archivos que había en el proyecto en *Eclipse* (siempre que se haya realizado bien el paso anterior). A su vez, *Jenkins* detectará que se ha producido un cambio en *Github*, y compilará el código que se encuentre en el repositorio, además de guardarlo en su *workspace*. Se puede ver como se han producido estas acciones en las figuras 43 y 44.

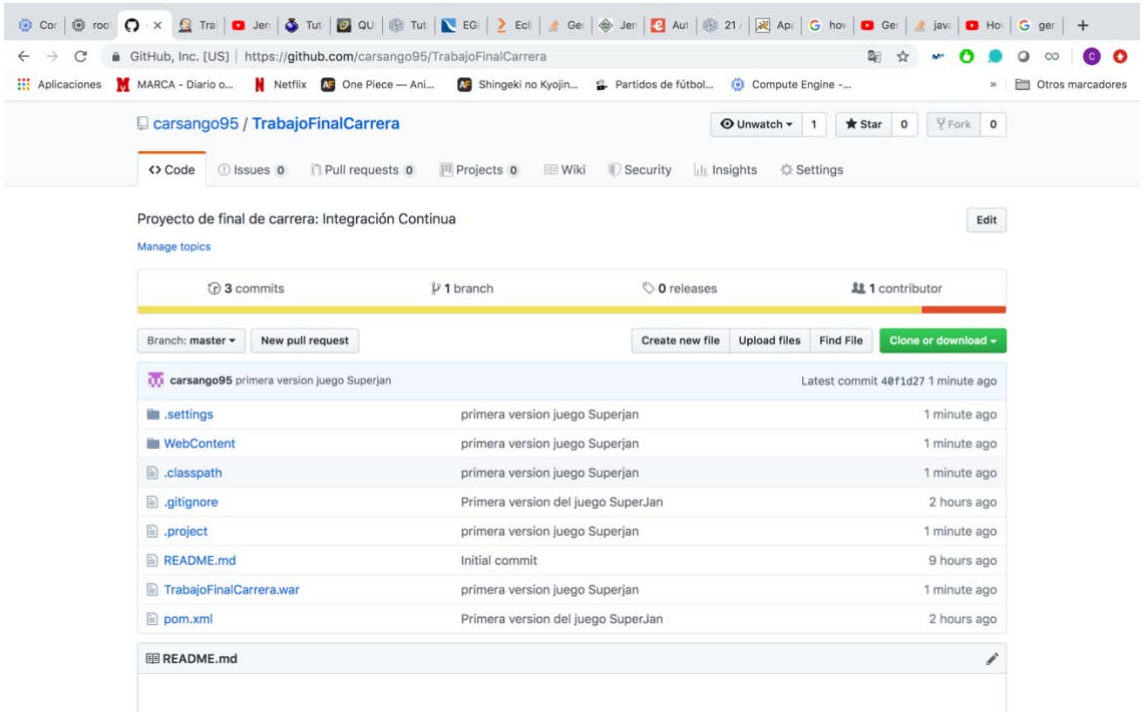


Figura 43. Actualización de repositorio en Github (Elaboración propia)

```
root@instance-2:/opt/tomcat/.jenkins/workspace/TrabajoFinalCarrera# ls
pom.xml  README.md  TrabajoFinalCarrera.war  WebContent
```

Figura 44. Actualización del workspace de Jenkins en el servidor en línea (Elaboración propia)

4. Configuración del despliegue continuo y primer despliegue

Una vez realizado el primer *commit*, hay que configurar el despliegue automático de la aplicación web que se está desarrollando. Para ello, hay que indicarle a *Jenkins* cuales son nuestras credenciales de acceso a *Tomcat* para que, automáticamente, acceda a este y despliegue la aplicación. Para ello, en el panel de control inicial de *Jenkins*, hay que clicar en *Credentials* → *System* → *Global credentials (unrestricted)*. Aquí se pueden incluir credenciales que más tarde se podrán usar para sincronizar

Jenkins con otras herramientas (en el caso que nos ocupa, con *Tomcat*). El nombre de usuario y contraseña que se deben escribir son los que se establecieron en el apartado de configuración de *Tomcat* (los mismos que se pueden ver en la figura 20). En la figura 45 se puede ver un ejemplo.

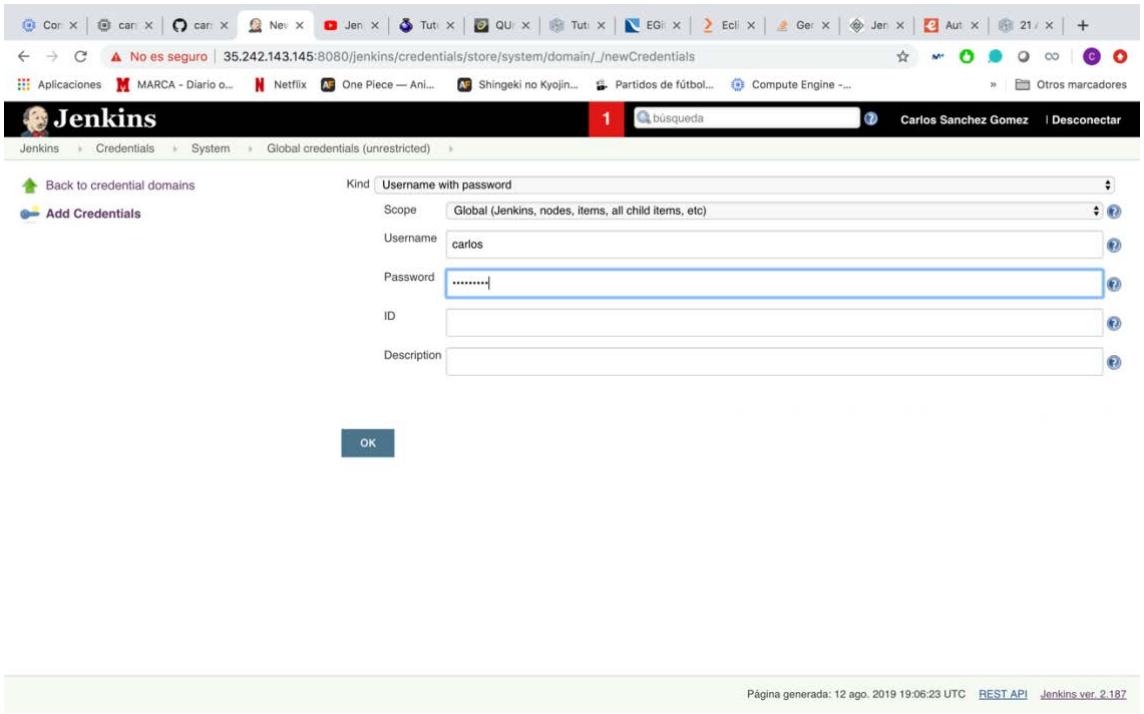


Figura 45. Creación de credenciales de Tomcat en Jenkins (Elaboración propia)

Una vez guardadas las credenciales de *Tomcat*, se debe configurar el despliegue en *Jenkins*. Para ello, En el proyecto que se creó en *Jenkins*, hay que clicar en *Configurar*. Se abrirá la ventana de configuración que ya se modificó cuando se creó el proyecto. Aquí, si se instaló correctamente en su momento el *plugin* "Deploy war/ear to a container", se debería poder ver en el desplegable de "Acciones para ejecutar después" dicha opción (ejemplo en figura 46). Seleccionada esta opción, se abrirá una ventana en la que se tiene que especificar primero el *container* que utilizaremos (en nuestro caso, *Tomcat 9*). También se debe especificar el nombre del archivo que se desplegará, las credenciales de *Tomcat*, y la *URL* de nuestro *Tomcat*. Debe quedar todo tal y como se ve en la figura 47 (recordando siempre utilizar la IP de nuestro servidor, y seleccionar las credenciales que anteriormente se configuraron en *Jenkins*).

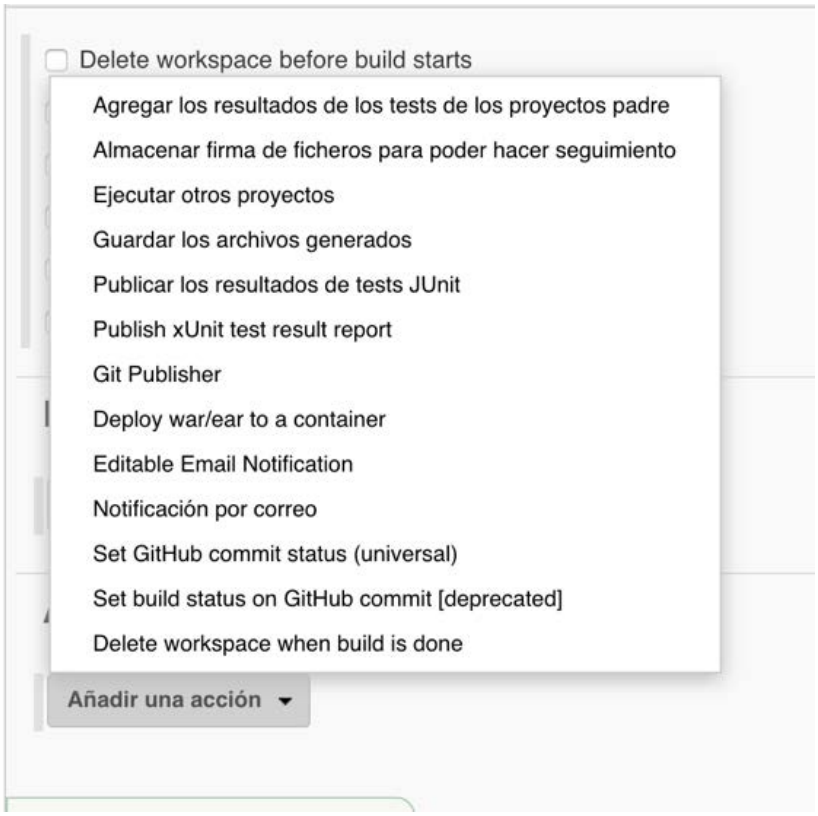


Figura 46. Configuración de despliegue en Jenkins (Elaboración propia)

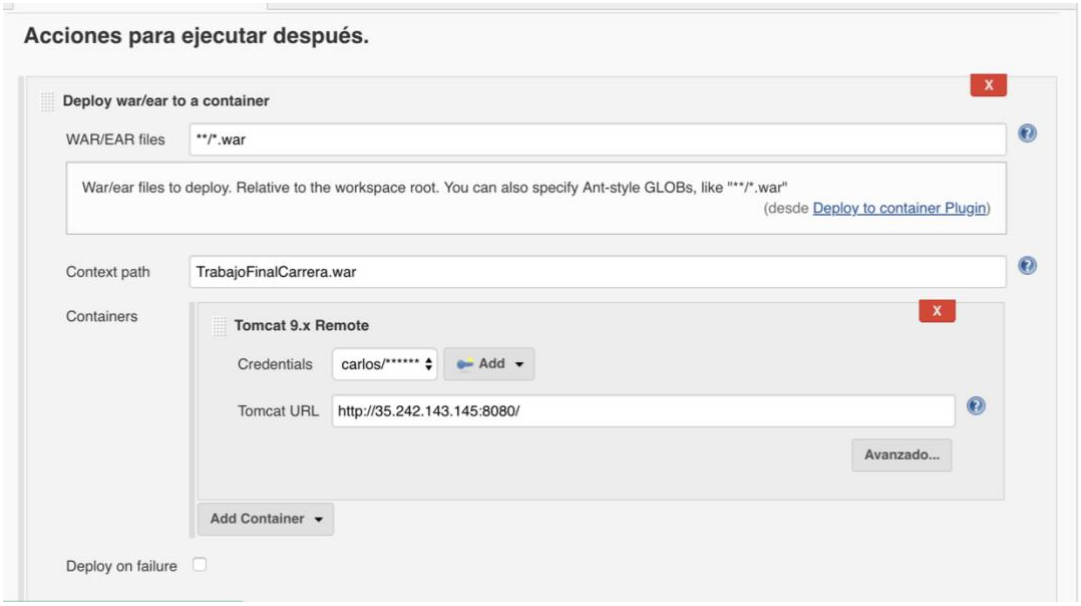
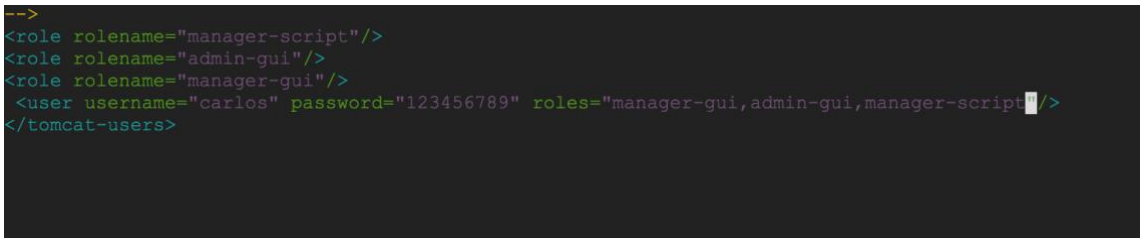


Figura 47. Configuración de despliegue en Jenkins (Elaboración propia)

Paralelamente, se debe realizar una modificación en el fichero *tomcat-users.xml*, pues para poder desplegar, se debe añadir un rol más al usuario que se creó en *Tomcat*. El nombre de dicho rol es *manager-script*; a continuación se recuerda el comando para modificar este archivo, en la figura 48 podemos ver un ejemplo de cómo debe quedar modificado dicho archivo.

```
sudo nano /opt/tomcat/conf/tomcat-users.xml
```




```
<role rolename="manager-script"/>
<role rolename="admin-gui"/>
<role rolename="manager-gui"/>
 <user username="carlos" password="123456789" roles="manager-gui,admin-gui,manager-script"/>
</tomcat-users>
```

Figura 48. Modificación del archivo tomcat-users.xml (Elaboración propia)

Con estas modificaciones, *Jenkins* ya está preparado para realizar despliegues automáticos, lo que no quita que el usuario, si lo necesitase, puede realizar un despliegue manual. *Jenkins* realizará un despliegue cada vez que se realice una compilación de código, por lo que, si se quisiese realizar un despliegue manual, basta con realizar una compilación manual. Para ello, habrá que clicar, dentro del panel de control del proyecto, en la opción de *Construir ahora* en la barra izquierda de opciones.

Realizada la compilación (ya sea manual o automática), en la parte inferior izquierda del panel de control de *Jenkins*, se puede ver un historial con todas las compilaciones que se han realizado, nombradas con el número de compilaciones que se han realizado. Si aparece con color azul, significa que no ha habido errores. Si aparece en rojo, significa que ha ocurrido algún error. Para acceder a la consola y ver que ha pasado con la compilación, basta con clicar en el número de la compilación en el historial. En ella se puede comprobar si se ha realizado la compilación y el despliegue. SI todo ha ido correctamente, debería aparecer un *log* parecido al que se muestra en la figura 49.

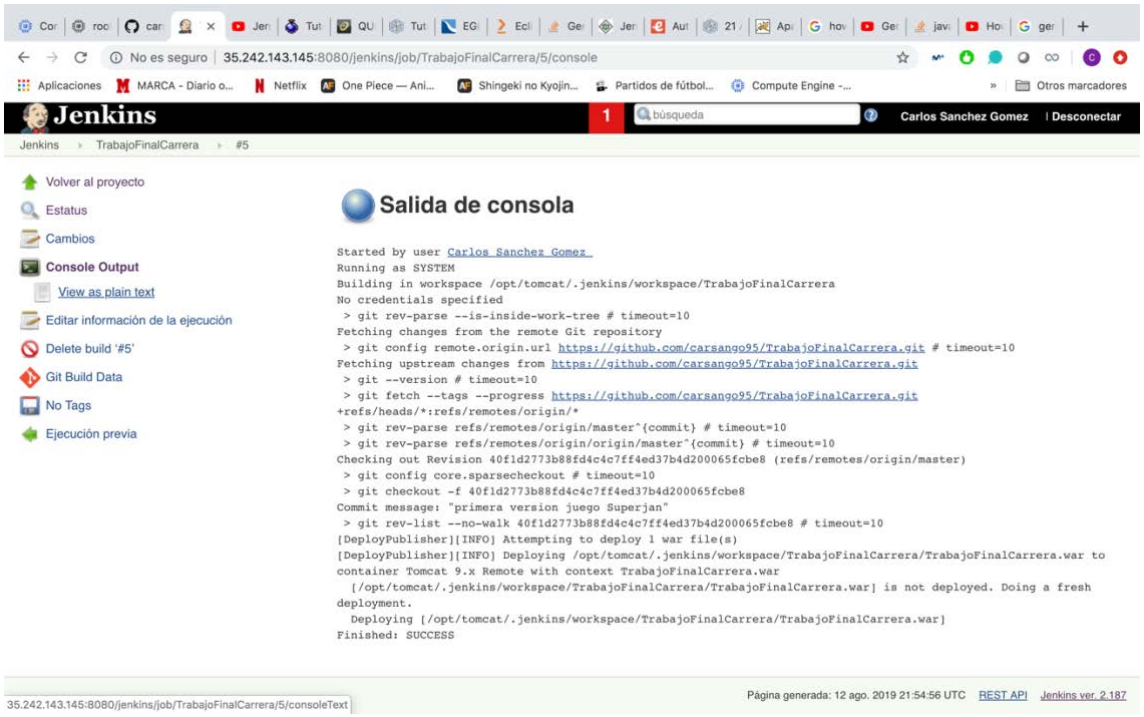


Figura 49. Log de compilación en Jenkins (Elaboración propia)

En el *log*, se puede ver que el *deploying* (despliegue) se ha realizado correctamente. Si se accede a la página *manager* de *Tomcat*, en la que se pueden ver todas las aplicaciones que el servidor web tiene desplegadas, se puede apreciar que la aplicación del proyecto que nos ocupa, llamada *TrabajoFinalCarrera.war*, se encuentra desplegada (ver figura 50). Si se clica en dicha aplicación, se abrirá una ventana del navegador, en la que, como se puede comprobar en la figura 51, se ha ejecutado correctamente la aplicación.

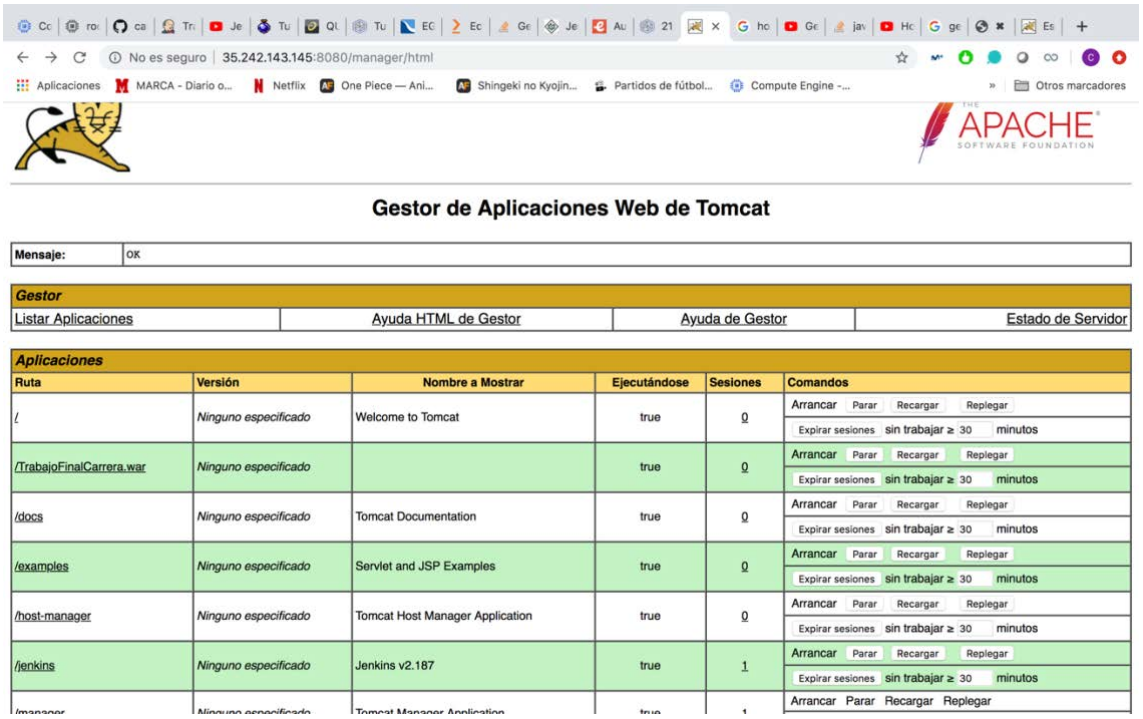


Figura 50. Sección manager de Tomcat (Elaboración propia)

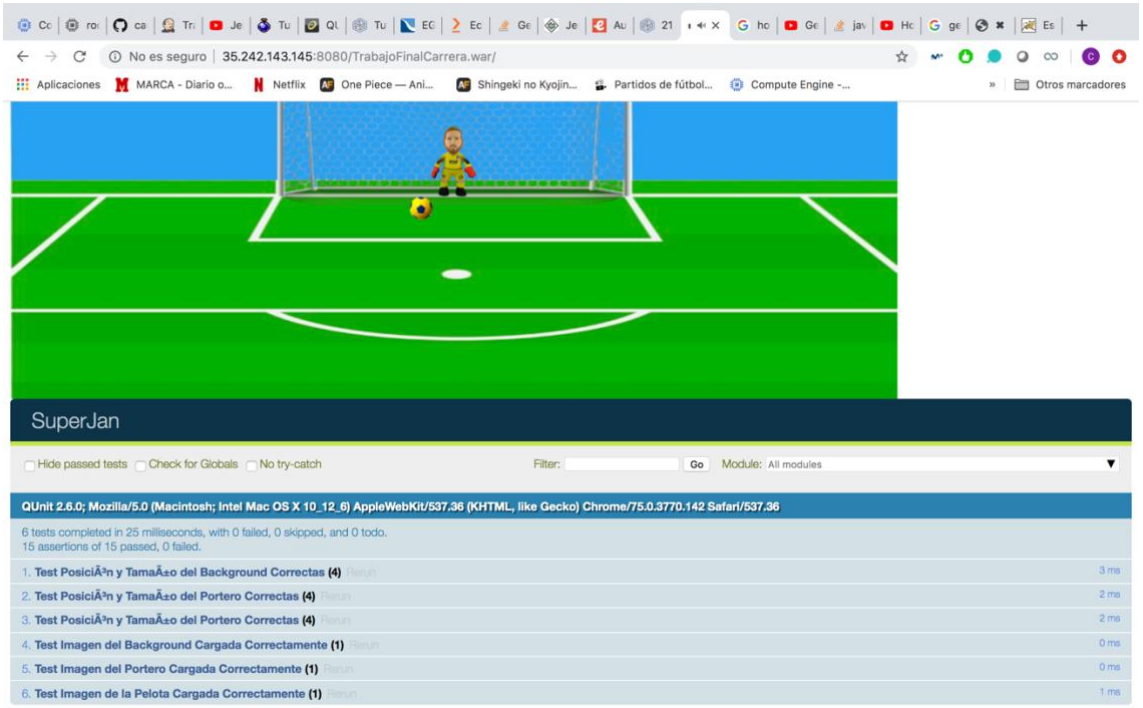


Figura 51. Aplicación web ejecutada en el navegador a través de Tomcat (Elaboración propia)

5. Segunda implementación de código, integración y despliegue

Una vez desplegada la primera versión de la aplicación, comenzaría el segundo *sprint*, es decir, se comenzaría a desarrollar la segunda versión, que, como se explicó en la descripción de esta, mejoraría la anterior, adaptando la velocidad de los balones para que mejore la jugabilidad, y haciendo que estos aparezcan solo en dirección a la portería.

El proceso a seguir será el mismo que en el primer *sprint*: el desarrollador modificará el código en local en *Eclipse*, y lo ejecutará para comprobar el resultado de las pruebas (figura 52), exportará el proyecto en forma de *.war*, y lo meterá dentro del proyecto en *Eclipse*, para acto seguido realizar el *commit* con los cambios realizados (figura 53).

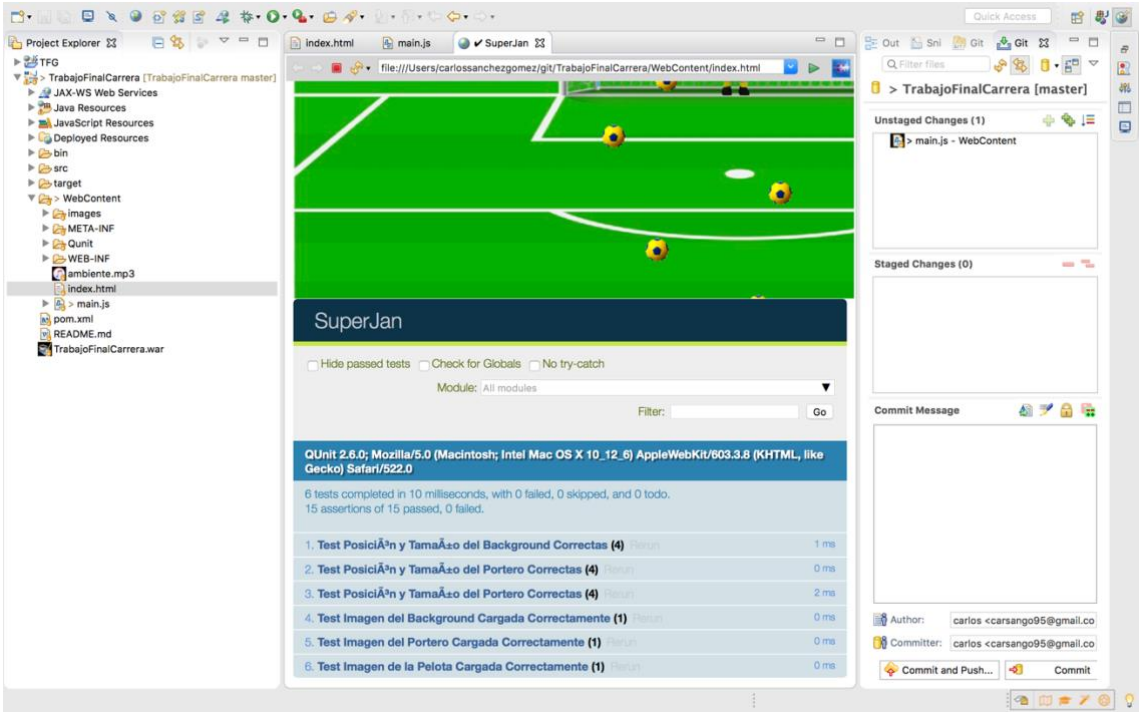


Figura 52. Ejecución de pruebas en Eclipse (Elaboración propia)

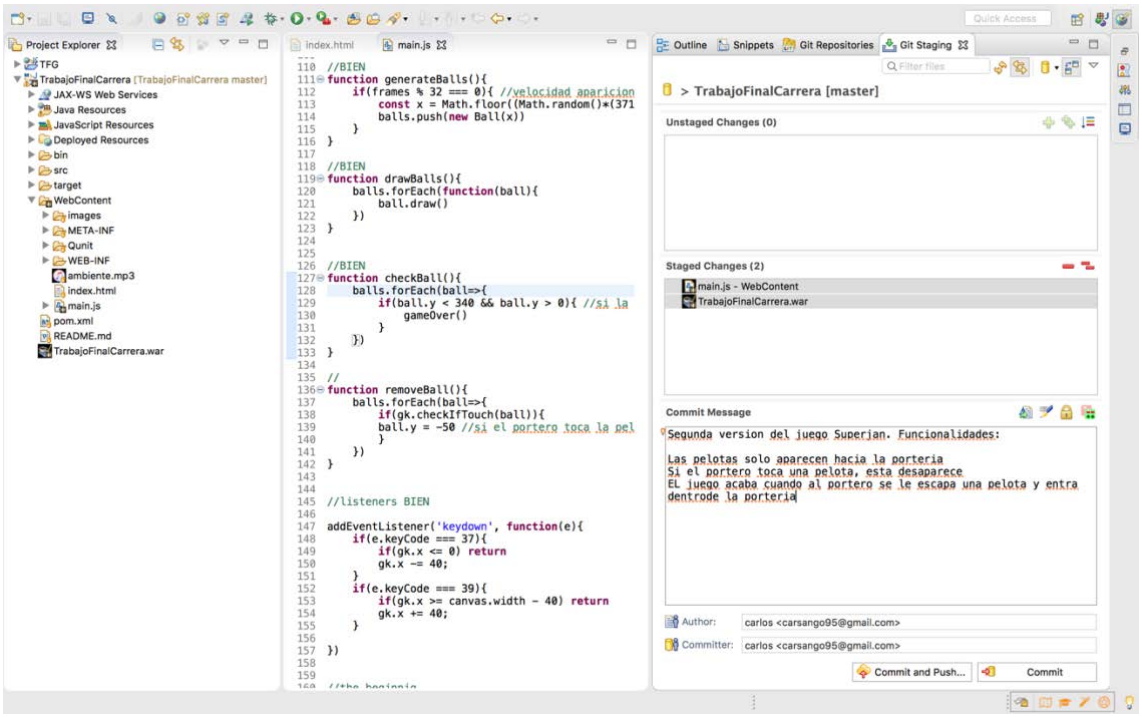


Figura 53. Realización de commit de la segunda versión de la aplicación (Elaboración propia)

Hecho el *commit*, el resto de las acciones se realizarán automáticamente. Como se puede apreciar en las figuras 54 y 55, en *Github* se ha actualizado el repositorio con los archivos que se han modificado (aquellos que tienen en la descripción el mensaje que se ha añadido en el segundo *commit*), y en *Jenkins* se ha realizado una compilación y despliegue (el color azul nos indica que todo ha ido correctamente), como se puede ver en el *log* de la consola.

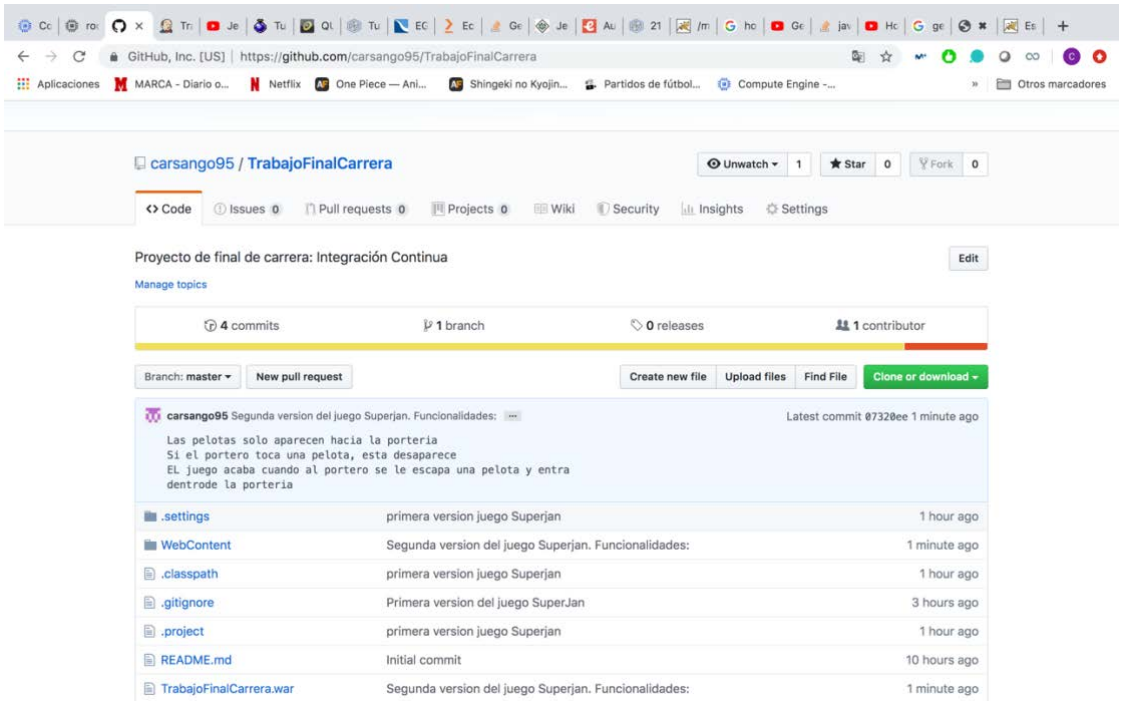


Figura 54. Actualización del repositorio de Github con el segundo commit (Elaboración propia)

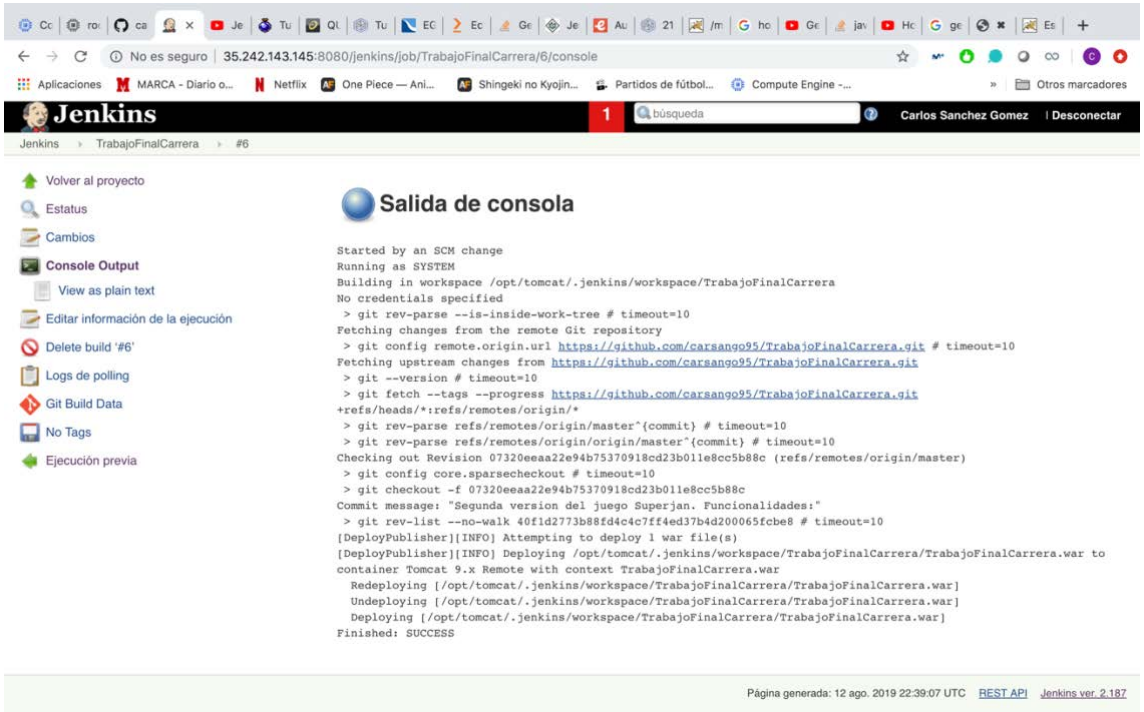


Figura 55. Compilación y despliegue automáticos en Jenkins (Elaboración propia)

6. Tercera implementación de código, integración y despliegue

Desplegada la aplicación tras el segundo *commit*, comenzará el tercer y último *sprint* del proyecto, en el que se mejorará la aplicación, otorgándole tres oportunidades o vidas al usuario, y añadiendo un marcador al final de la partida, que recoge el número de pelotas que el usuario pudo salvar.

Se seguirá el mismo *workflow* que en los anteriores *sprints*:

-Implementación de código y ejecución de pruebas en *Eclipse* (figura 56)

-Exportación del proyecto en *.war* y realización del tercer *commit* (figura 57)

-El repositorio de *Github* se actualiza automáticamente (figura 58), *Jenkins* detecta cambios y compila (figura 59) y despliega en *Tomcat* (figuras 60 y 61).

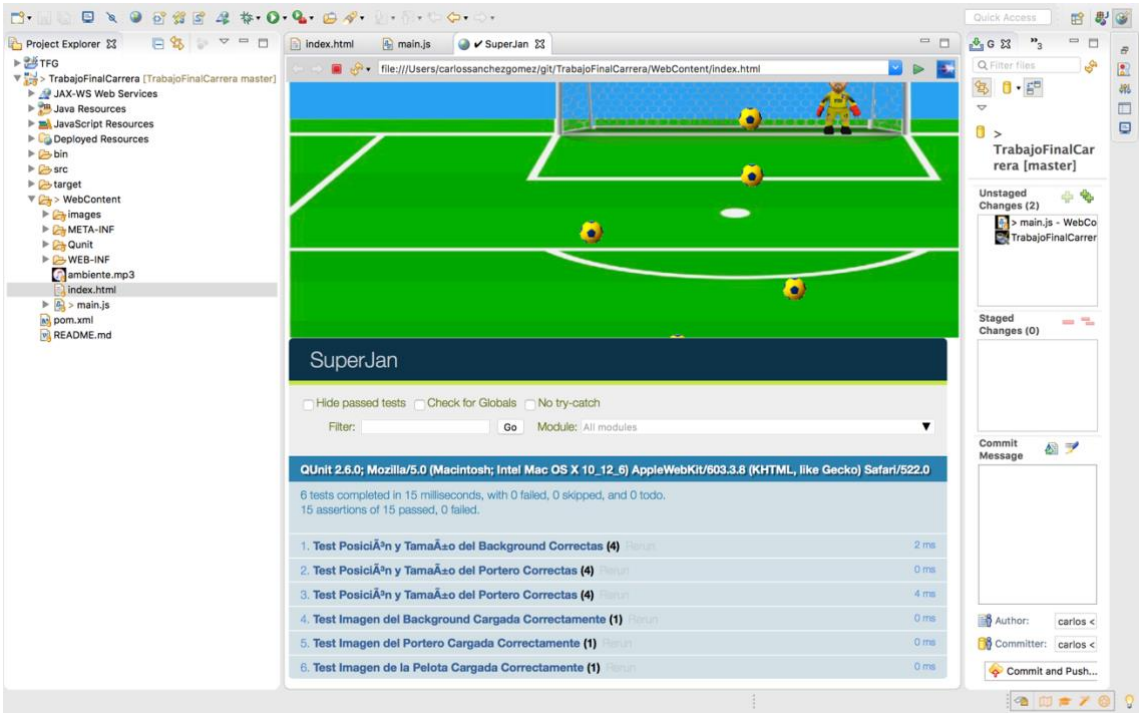


Figura 56. Ejecución de pruebas en Eclipse (Elaboración propia)

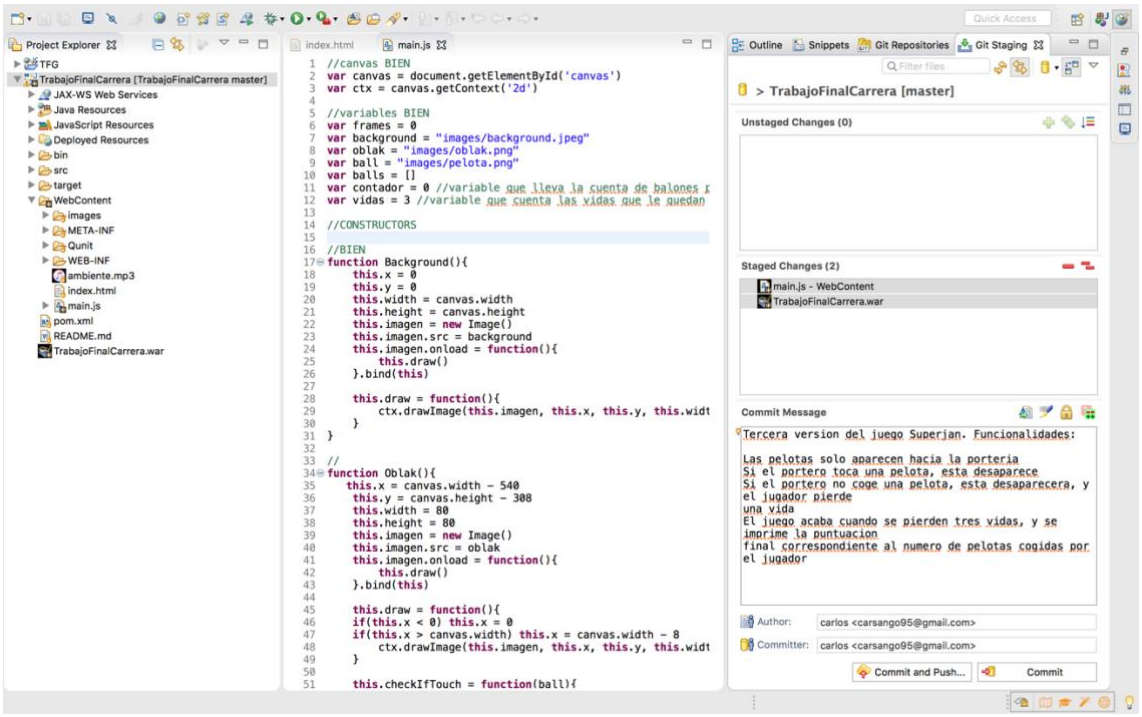


Figura 57. Realización del tercer commit (Elaboración propia)

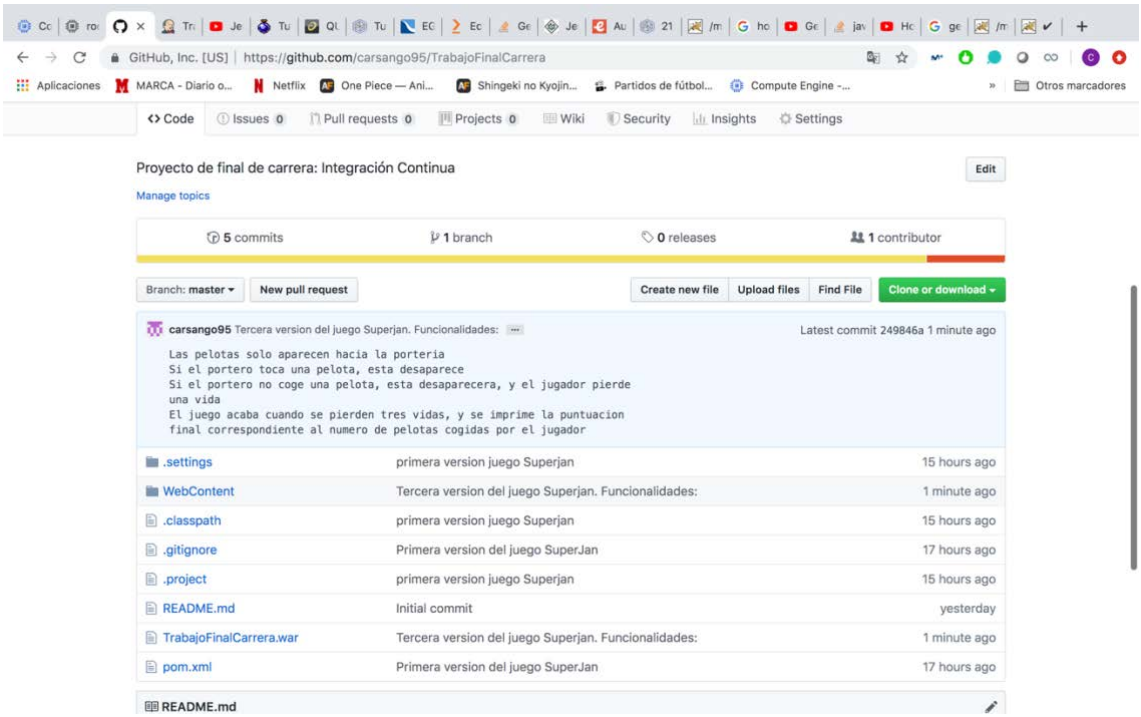


Figura 58. Actualización del repositorio de Github (Elaboración propia)

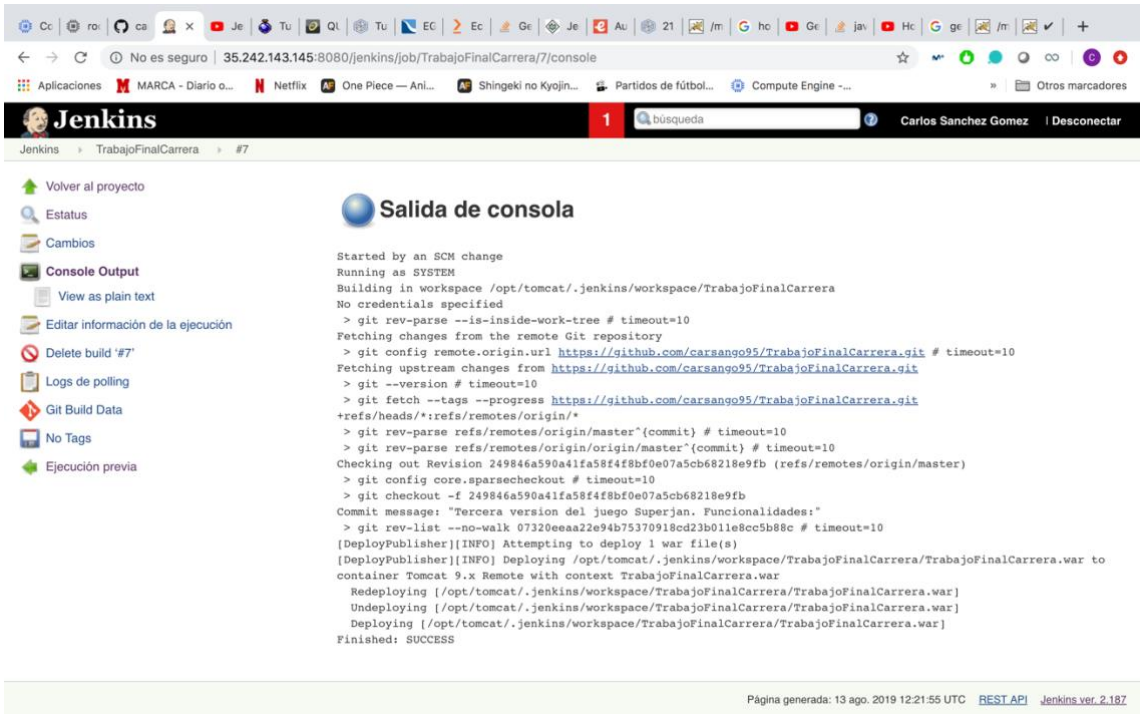


Figura 59. Compilación y despliegue automáticos en Jenkins (Elaboración propia)

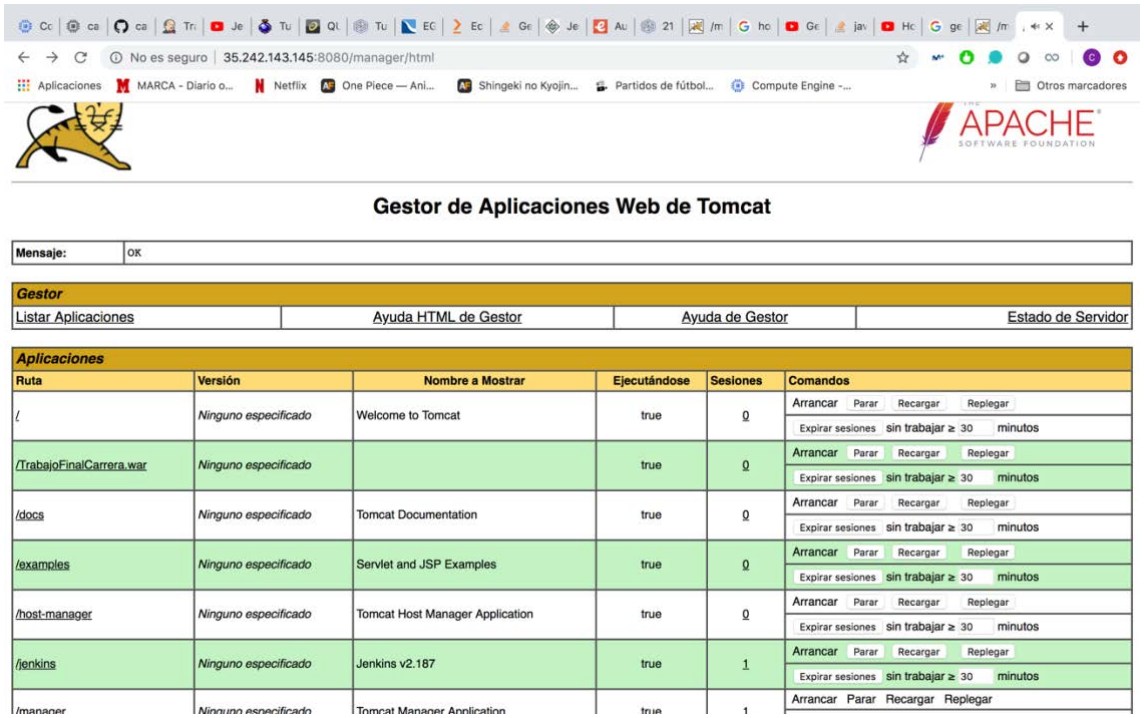


Figura 60. Aplicación desplegada en Tomcat (Elaboración propia)

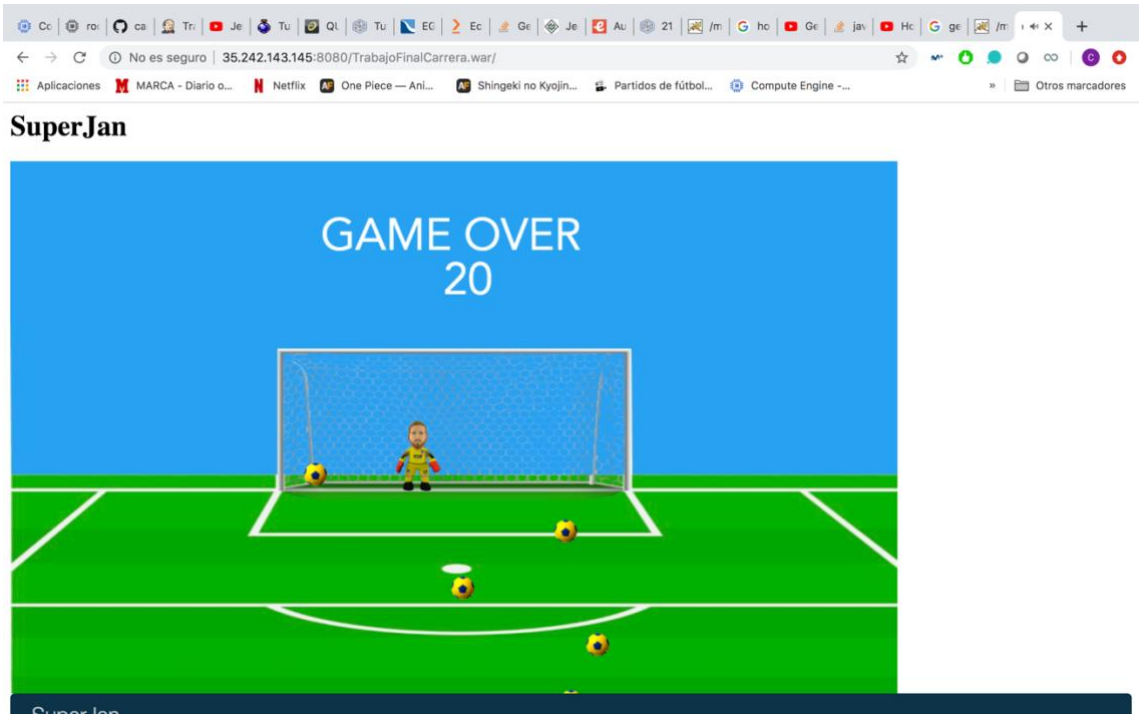


Figura 61. Aplicación ejecutada a través de Tomcat (Elaboración propia)

III. LA AGILIDAD EN EL MUNDO DE LA EMPRESA. ANÁLISIS DE EFICIENCIA DE EMPRESAS ÁGILES.

A. REPASO DE LA LITERATURA PREVIA

En este apartado, se expondrán todas las características de la metodología *Agile* que están relacionadas con el mundo de la empresa, así como un par de ejemplos en los que dos grandes empresas como *Orange* y *BBVA* han conseguido implementar con éxito esta metodología de trabajo, y han obtenido resultados positivos, no sin antes realizar una introducción acerca de qué se entiende por innovación, y su aplicación en la empresa.

1. ¿Qué es la innovación?

El concepto de innovación es muy amplio, son muchos los autores y estudiosos que, a lo largo de la historia, han intentado crear una definición concreta acerca de qué es la innovación. Por ejemplo, André Piatier, economista y estadístico francés, definió en 1987 la innovación como *una idea transformada en algo vendido y usado* (Escorsa & Valls, 2003). En el Manual de Frascati, redactado por la OCDE en 1963, se especifica como *la transformación de una idea en un producto o servicio comercializable, un nuevo procedimiento de fabricación, o de proporcionar un servicio social* (Escorsa & Valls, 2003). Se puede encontrar otro ejemplo en la definición que proporciona el Departamento de Comercio e industria del Reino Unido: *la innovación es el proceso de adopción de una idea para satisfacer a los clientes de una forma efectiva y rentable* (Escorsa & Valls, 2003).

Todas estas definiciones tienen un factor común: entienden la innovación como la acción de transformar una idea novedosa en realidad, llevándola a la práctica (Escorsa & Valls, 2003). Sin embargo, como explica el economista británico Christopher Freeman, nunca habrá innovación si, al llevar dicha idea a la realidad, esta no resulta con una introducción fructífera en el mercado, es decir, con beneficios (Escorsa & Valls, 2003).

Una vez entendido el concepto de innovación, se diferenciará del concepto de **innovación tecnológica**. La tecnología no es otra cosa que la *aplicación sistemática del conocimiento científico u otro conocimiento organizado a tareas prácticas* (Fernández

Sánchez & Vázquez Ordás, 1996; Galbraith, 1980), por lo que la innovación tecnológica será toda aquella innovación que implique el uso de la tecnología, mediante la introducción de algún tipo de cambio tecnológico en el producto, o en los procesos (Escorsa & Valls, 2003).

En el proceso de innovación tecnológica, en el que las empresas buscan generar beneficios mediante la aplicación de ideas tecnológicas y novedosas, es donde entra la práctica de las actividades de **I+D**, siglas de los términos "investigación" y "desarrollo" (Escorsa & Valls, 2003; Fernández Sánchez & Vázquez Ordás, 1996). Básicamente, el I+D es lo que usan las empresas para innovar.

Mediante la **investigación**, se busca obtener conocimiento nuevo; puede ser de dos tipos: **básica**, si se busca conocimiento científico nuevo para la formulación de leyes, hipótesis o teorías, sin ningún tipo de fin lucrativo, o **aplicada**, que también busca conocimiento científico nuevo, pero más enfocados a un fin práctico, y se pueden llegar a servir de los resultados obtenidos a través de la investigación básica (Escorsa & Valls, 2003; Fernández Sánchez & Vázquez Ordás, 1996).

Una vez realizada la labor de investigación, en la cual conseguimos la idea innovadora (ya sea un nuevo producto, una modificación de uno ya existente, o innovaciones en procesos), empieza la fase de **desarrollo**, en la que se intentará conseguir que dicha idea pueda ser aplicada de forma práctica para su posicionamiento en el mercado, y que se obtenga un beneficio a través de su aplicación (Escorsa & Valls, 2003; Fernández Sánchez & Vázquez Ordás, 1996; OCDE, 2003)

2. Innovación en la empresa

Todo este proceso conforma lo que se conoce como innovación tecnológica. Esta actividad resulta crucial en el mundo de la empresa, pues es un mundo en constante movimiento, en el que los cambios están a la orden del día, y el que no se adapta a ellos, pierde competitividad y, por ende, puede verse abocado al fracaso (WinLead, 2018). Las empresas se sirven de dos tipos diferentes de innovación, con dos enfoques totalmente distintos (Christensen, 1997): la **innovación incremental**, mediante la cual la empresa realiza cambios graduales a lo largo del tiempo, intentando mejorar progresivamente productos, procesos y/o servicios, y la **innovación disruptiva**, en la que se busca realizar

un cambio brusco en el mercado, para situarse como empresa líder en el mercado, y forzando al resto de competidores a modificar su enfoque de negocio (WinLead, 2018).

La importancia de la innovación dentro de una empresa ha sido estudiada a lo largo de la historia; cabe destacar al famoso economista Joseph A. Schumpeter (fue ministro de finanzas austriaco y profesor de economía en Harvard), el cual dedicó gran parte de su trabajo al estudio de la empresa como principal motor de desarrollo económico (Montoya Suárez, 2004). Schumpeter describió la empresa como aquel negocio que se sirve de la innovación para obtener beneficios, y las diferencia de aquellos que se basan en un modelo asentado, en periodo de madurez, y que reniegan de la innovación, a los que no les otorga el calificativo de "empresa" (Schumpeter, 1996; Montoya Suárez, 2004). También existen organizaciones que han dedicado tiempo al estudio de la innovación, su importancia, y aplicación, como la OCDE (Organización para la Cooperación y Desarrollo Económicos), que define la innovación como la "introducción de un nuevo, o significantemente mejorado, producto de un proceso, de un nuevo método de organización o de un nuevo método organizativo, en las prácticas internas de la empresa, la organización del lugar de trabajo o las relaciones exteriores" (OCDE, 2006).

Una vez una empresa ha conseguido introducirse en el mercado y tener su cartera de clientes, existen dos enfoques diferentes para el futuro: la **supervivencia empresarial**, o la búsqueda de una **nueva ventaja competitiva** (Naranjo, 2016). La supervivencia empresarial se centra en mantener el producto que les ha otorgado esa cuota de mercado tal y como lo concibieron, y basar todos sus ingresos en dicho producto, mientras que el otro enfoque utiliza la innovación para conseguir generar una ventaja competitiva que permita desmarcar a la empresa de la competencia y situarla como líder en el mercado (Naranjo, 2016; Alfaro Giménez & Pina Massachs, 2018). La generación de dicha ventaja se producirá mediante la creación o adición de valor a la cadena de valor del producto, servicio o proceso de la empresa, añadiendo una característica única que permita desmarcarse de la competencia (Alfaro Giménez & Pina Massachs, 2018).

La razón por la que una empresa toma (o debería tomar) la decisión de innovar es simple. Si una empresa sigue el enfoque de supervivencia empresarial, a corto plazo puede conseguir beneficios, pues tiene cierta cuota de mercado. Sin embargo, en cuanto uno de sus competidores consiga generar una ventaja competitiva que le desmarque del resto, el producto de dicha empresa quedará obsoleto y, por lo tanto, perderá toda la cuota de mercado que tenía. Ejemplo de esta situación es el caso Kodak o el caso Nokia que,

siendo lideres en el mercado de las cámaras fotográficas y de los teléfonos móviles, respectivamente, decidieron no innovar, y en cuanto aparecieron otras marcas que sí tomaron dicha decisión, no solo perdieron cuota de mercado, sino que prácticamente desaparecieron del mismo (Naranjo, 2016). Esto son solo dos ejemplos de la multitud de casos de empresas que decidieron no innovar y en cuestión de años desaparecieron del mercado, y que reflejan la importancia de la innovación en el mundo de la empresa.

Una vez explicado el concepto de innovación, y la importancia del mismo, la siguiente pregunta que es necesario responder es la siguiente: ¿Cómo se puede innovar en una empresa? Para introducir innovación en una empresa (ya sea mediante la creación de un producto o servicio nuevo, o la introducción de nuevos métodos de organización en la misma), lo primero y más esencial, es conseguir adaptarse a las siguientes tres características (WinLead, 2018):

- **Creatividad**: que permita encontrar soluciones originales y diferentes a las ya existentes en el mercado.

- **Valentía**: que permita dar un paso al frente y arriesgar para conseguir esa diferenciación buscada.

- **Conocimiento**: que permita saber que necesita o quiere el cliente, y en qué momento.

Siguiendo estas tres máximas, una empresa puede conseguir el éxito a través de la innovación, si bien la innovación es un proceso complejo, que consta de varias fases, a cada una de las cuales debemos prestar mucha atención. Estas fases las podemos encuadrar en cuatro bloques diferenciados (Asociación de la Industria Navarra, 2008):

- **Dimensión estratégica:** En esta fase, la empresa debe realizar un Plan Estratégico Tecnológico (**PET**), que no es otra cosa que la organización del proyecto de innovación, mediante un **diagnóstico previo** de la situación actual respecto de la tecnología que se quiere utilizar (necesidades o gustos del cliente, tecnología actual…), a partir del cual **definiremos las estrategias** que

sigue la empresa con respecto a la tecnología actual, y las que deberíamos implantar para conseguir la tecnología deseada**,** para así poder marcar el **camino a seguir** para cambiar de la tecnología utilizada actualmente, a la deseada (Asociación de la Industria Navarra, 2008).

- **Identificación de ideas a desarrollar:** La creatividad es la base de esta fase, en la que se crearan todas las ideas que fundamentaran el proceso de innovación. Lo primero será **plantearse el problema** en cuestión, a partir del cual se empezarán a **lanzar ideas** que puedan servir para solucionarlo. Las ideas se lanzarán de forma intuitiva, pues el siguiente paso será **analizar** cual de dichas ideas cumplen real y racionalmente el problema planteado, para terminar analizando si aquellas que podrían solucionar dicho problema, son realmente **viables** a vistas de implantación y desarrollo en la empresa (Asociación de la Industria Navarra, 2008).

- **Desarrollo de los proyectos:** Esta fase se divide en dos procesos esenciales: la gestión y la financiación del proyecto. En la **fase de gestión**, se debe, como primera acción, definir objetivos y recursos del proyecto. Seguidamente viene la fase de planificación, en la que se determinarán las tareas a realizar por los equipos para la consecución del proyecto, para después pasar a la fase de ejecución y control, en la que se realizarán (y supervisarán) estas tareas, y la de finalización, donde se recopilarán posibles mejoras futuras para la reciente implantación. En la **fase de financiación**, se analizará cuales son las posibles y mejores fuentes de financiación para el proyecto de innovación, que pueden ser subvenciones, créditos, préstamos participativos, entre otros (Asociación de la Industria Navarra, 2008).

- **Explotación de los resultados:** Esta última fase consta de dos acciones que son esenciales para el cierre con éxito de la implantación de un proyecto de innovación. La primera sería el **aseguramiento de la innovación**, el cual conseguiremos a través de la Propiedad Industrial, protegiendo, mediante la ley, la innovación en el producto, servicio o proceso que hemos implantado. La segunda se refiere a la **explotación o rentabilidad de la innovación**, que

consiste en la recuperación de la inversión realizada para la implantación de la innovación (Asociación de la Industria Navarra, 2008).

3. Ejemplos de innovación en grandes empresas

Vista la importancia de la innovación en la empresa, es normal que se puedan encontrar ejemplos muy representativos de innovación en grandes empresas de cualquier sector. En este epígrafe, se expondrán casos de innovación en dos empresas del Ibex 35 de diferentes sectores: **Telefónica SA** (sector tecnología y telecomunicaciones) y **Santander SA** (sector financiero).

3.1 Telefónica SA

Telefónica es, a nivel nacional, una de las empresas mejor posicionadas en el mercado (en términos de facturación anual), siendo la mejor posicionada en el sector tecnología y telecomunicaciones (El Economista, 2019). Siendo la empresa líder en este sector, es entendible que su inversión en I+D sea de las más importantes en lo que al sector nacional se refiere, alcanzando los 947 millones de euros en 2018, y quedando detrás únicamente del Banco Santander; en Europa, es la segunda operadora con mayor inversión en I+D del mercado (Millán Alonso, 2019).

Toda esta inversión en desarrollo tecnológico se puede ver reflejada en los numerosos proyectos que Telefónica está llevando a cabo, tanto de creación de nuevos productos y procesos, como de actualización de los ya existentes. A continuación, se expondrán algunos ejemplos de los mismos (proyectos a los que fue destinada la inversión en I+D del año 2018).

Telefónica divide su rango de actividades en cuatro plataformas diferentes, estando la primera dedicada a las infraestructuras y redes, la segunda, al desarrollo de aplicaciones, la tercera, a productos y servicios, y la cuarta, dedicada íntegramente a la inteligencia artificial (Millán Alonso, 2019). En lo referente a la **primera plataforma**, la operadora invirtió tanto en el desarrollo de la actual tecnología *4G*, como en el de la aun en pruebas tecnología *5G*, y en otros proyectos como el llamado *Internet para Todos*, el cual pretender llevar internet a más de 100 millones de personas en Latinoamérica (Millán Alonso, 2019).

En cuanto a la **segunda y tercera plataforma**, Telefónica ha impulsado, gracias a la inversión en I+D, el desarrollo, tanto de sus aplicaciones móviles (para continuar con su proceso de digitalización), como de su servicio de video en streaming, *Movistar +*, uno de los líderes en el sector de plataformas de video en streaming (Millán Alonso, 2019).

La **cuarta plataforma**, la más novedosa y recientemente implementada, es una de las apuestas de futuro de Telefónica, está dedicada íntegramente al desarrollo de un nuevo asistente virtual, *Aura*, mediante el cual Telefónica pretende adentrarse en el mundo de la inteligencia artificial, el Big Data y el Machine Learning (Millán Alonso, 2019). Esta es una de las mayores inversiones en desarrollo de Telefónica, pues están intentando implementar una forma totalmente nueva y revolucionaria de dar uso a la inteligencia artificial, colocando al asistente virtual en cada uno de los servicios y productos que la compañía ofrece a sus clientes, como si fuera un asistente físico de la compañía, al cual puedes preguntarle cualquier cosa, y está disponible en el mismo momento en el que lo necesitas (Telefónica, 2017)

Además de las mencionadas plataformas, también hay que mencionar el modelo de desarrollo tecnológico que sigue Telefónica, basado en **startups internas**. Es un modelo innovador, en el que cada uno de los proyectos de I+D que Telefónica inicia, es desarrollado por un intraemprendedor, el cual es elegido de una lista de emprendedores que presentan ideas innovadoras, escogiendo al más prometedor de ellos, dándole financiación para permitirle desarrollar el proyecto. Todo este modelo está basado en la conocida metodología *Lean Startup*, siendo Telefónica pionera a nivel mundial en utilizar esta metodología en una gran empresa (Telefónica , 2016).

3.2 Santander SA

El Banco Santander es una de las entidades financieras más importantes a nivel nacional e internacional, superando al resto de entidades financieras españolas más importantes (en cuanto a valor de la marca se refiere), y compitiendo con las entidades financieras internacionales por el liderato del mercado financiero (MarketingNews, 2019). Si ha alcanzado ese nivel de liderazgo en el mundo de las financieras es, en parte, por la importante inversión que viene realizando el Santander en I+D, alcanzando en 2018 los 1468 millones de euros, situándole como primera empresa española y primer banco mundial en inversión de investigación y desarrollo (Santander SA, 2018; European

Commission, 2018). Toda esta inversión que el Banco Santander ha realizado en I+D se puede ver reflejada en los ejemplos que se expondrán a continuación.

Uno de los mejores ejemplos de inversión en investigación y desarrollo en el Banco Santander es el impulso que están dando a una de sus filiales, **Openbank**. Openbank es la filial digital del Banco Santander, en la que todas las transacciones se realizan digitalmente; solo disponen de una oficina física (Veloso, 2018). Openbank comenzó siendo el primer banco telefónico de España, y, desde su creación, siempre han apostado por la banca telemática, adaptándose siempre a la tecnología del momento (Openbank, 2019). En la actualidad, el Banco Santander sigue apostando por Openbank: la cartera de clientes ha crecido un 26%, y los depósitos un 20%, alcanzando los 9000 millones de euros (Veloso, 2018). Openbank se renueva constantemente, una de las últimas novedades incluidas en la filial digital del Santander es el llamado *"Roboadvisor"*, una herramienta que permite al cliente invertir en activos y fondos indexados de otras entidades, configurando las metas de ahorro e inversión deseadas, y con comisiones relativamente bajas, que oscilan entre el 0,55% y el 0,85% del capital que se ha invertido (Veloso, 2018). Otra de las novedades agregadas es la posibilidad de agregar el resto de cuentas que el cliente tenga en otros bancos, a la aplicación de Openbank, para poder realizar cualquier consulta necesaria acerca de cualquiera de ellas, en una misma plataforma (Veloso, 2018).

Otro campo en el que el Banco Santander ha hecho hincapié en invertir ha sido el de la **ciberseguridad**. En un mundo digitalizado casi completamente, se hace necesario velar por la seguridad de las carteras de los clientes, y de los propios activos financieros de las entidades. Es por ello por lo que, en el Santander, dedican gran parte de su inversión en I+D a la ciberseguridad, alcanzando (durante tres años, desde 2018) los 400 millones de euros (Armengol, 2018). Esto se ha visto en la creación de, no solo un departamento de ciberseguridad, sino un centro especializado en ella, con unos 170 empleados, situado en la Ciudad Financiera de Boadilla (centro de operaciones del Banco Santander), agrupando a ingenieros informáticos y hackers que se encargaran de la seguridad cibernética de la entidad (Cabrera, 2019).

Destacable es también la inversión que está realizando el Banco Santander en las conocidas ***Fintech*** *(Financial Technonlogy),* empresas emprendedoras dedicadas a la resolución de problemas del mundo financiero a través de la tecnología (Santander SA, 2018; Integra IT, 2018). La estrategia del Santander respecto a las *Fintech* es la creación

de un fondo de capital de unos 200 millones de euros, denominado *Santander InnoVentures*, mediante el cual el banco invierte en diversas empresas dedicadas a la tecnología financiera, creando así una sinergia en la cual las *Fintech* aportan e introducen nuevas tecnologías al banco, y este les proporciona financiación, así como la experiencia de trabajar con una gran corporación como el Santander (Santander SA, 2018). A finales de 2018, el banco contaba con una cartera de unas 20 *Fintechs* dedicadas a campos como el *Marketplace Lending* o la inteligencia artificial, entre otros (Santander SA, 2018).

4. Metodología Agile en la empresa

Como se ha visto en los anteriores ejemplos, existen muchas formas de introducir desarrollo e innovación tecnológica en una empresa. Sin embargo, en este trabajo se tratará la introducción de una novedosa metodología de trabajo que, en los últimos años, viene pisando fuerte. Se trata de la ***metodología Agile,*** en la que lo principal es la percepción de los gustos y necesidades del cliente, y la agilidad de adaptar la empresa a los cambios que ello pueda suponer, frente a la búsqueda de la mejor relación calidad/precio de las metodologías tradicionales de trabajo (Mestre Valdés, 2018). Un cambio de estas características en la metodología de trabajo de una gran empresa suele ser costoso (en términos de tiempo y dinero), pero, si algo hemos aprendido a lo largo de la historia, es que, el que no se adapta a los cambios, no sobrevive. Estamos ante una especie de "darwinismo digital" (Mestre Valdés, 2018), en el que una empresa que no hace uso de la tecnología para adaptarse a los cambios, desaparecerá, y el cambio que se torna necesario en estos tiempos es el de adaptarse a las necesidades de un cliente que cada vez se vuelve más exigente, y requiere productos con características muy específicas.

La importancia de la *metodología Agile* se puede ver reflejada en la gran cantidad de empresas que la usan: hasta un 70% de empresas españolas (la mayoría del Ibex 35) la utilizan regularmente, y un 20% la usa como metodología base, si bien la mayoría de todas estas empresas pertenecen al sector financiero, donde es más fácil y útil su implantación, frente al sector industrial (IDG, 2018). Otra prueba del éxito de la *metodología Agile* es el gran incremento de la demanda de puestos de trabajo relacionados con la misma (Project Manager, Agile Coach, Scrum Master…) en los últimos años; las empresas han aumentado en un 50% la demanda de trabajadores especialistas en *Agile* (Galiana, 2019).

Si grandes empresas y corporaciones están implementando la *metodología Agile* en su forma de trabajo, se debe a los beneficios y **ventajas** que esta genera en ellas, como, por ejemplo, el aumento de la **velocidad de trabajo**, gracias a que esta metodología parte la entrega final del producto en entregas parciales, lo que permite detectar y corregir errores antes de la prueba final del producto, dándole más dinamismo a la cadena de producción (Galiana, 2019). Esta metodología tiene como premisa estar **centrada en el cliente**, lo que significa que la satisfacción del cliente es el principal objetivo del proyecto, llegando a considerar a este como a "uno más del equipo", participando en reuniones diarias, y pudiendo ver en todo momento el estado actual del proyecto. Teniendo en cuenta que, en la actualidad, la satisfacción de las necesidades del cliente es el factor que desequilibra la balanza, centrarse en el cliente es la mejor opción para la empresa (Machuca, 2018; Gil, 2019).

Otra de las ventajas de esta metodología es que, a diferencia de los modelos tradicionales, en los que existía una jerarquía con muchos escalones en la organización del proyecto, en la *metodología Agile* **la directiva mantiene una postura cercana** con respecto a los equipos del proyecto, llegando a trabajar codo con codo con ellos, lo que genera sensación de cercanía y compañerismo con respecto a los altos cargos en los miembros de los equipos, y también facilita la transferencia de *feedback* de los jefes hacia sus subordinados, y viceversa (Gil, 2019).

Otro beneficio de la *metodología Agile* es que otorga a la empresa la capacidad de **reaccionar a los cambios** del día a día, ya no solo en lo que respecta a los gustos y necesidades del cliente, sino a cambios tecnológicos que puedan surgir espontáneamente, y a los cuales es necesario adaptarse para no perecer. Esta metodología, al igual que el resto, conlleva una planificación inicial del proyecto, en la que se definen las acciones que se llevarán a cabo a lo largo del proyecto; sin embargo, cada cierto tiempo se realizan reuniones en las que se delibera para redefinir dichas acciones y la orientación de las mismas, en caso de ser necesario, para tener en cuenta todos los cambios mencionados anteriormente en el desarrollo del proyecto (Machuca, 2018).

Vistos los beneficios que la *metodología Agile* puede generar en una empresa, quedaría, por último, describir como sería el proceso de **transición** de una empresa que quiera adoptar dicha metodología. Para hacer que una empresa sea ágil, son tres los puntos que debemos cambiar: la **estructura**, o como formar equipos en cualquier nivel de la empresa; la **gobernabilidad**, que implica la toma de decisiones, la forma de trabajo, o

coordinación entre equipos (entre otros); **herramientas y métricas** que se utilizarán para medir el desempeño (Novoseltseva, 2018). Para realizar estos cambios, lo mejor es contratar a un profesional en esta metodología, los llamados ***Agile Coach***, cuyo trabajo es transmitir los principios y forma de trabajar de la *metodología Agile* en los empleados (Novoseltseva, 2018). Ya sea una pequeña o gran organización, y aunque ya existan empleados o directivos con conocimientos de esta metodología, la mejor opción es contratar un *Agile Coach*, pues no es lo mismo aplicar los conocimientos ágiles en tu trabajo, que intentar cambiar la forma de trabajar y organizarse de una organización, ya sea pequeña o grande (Novoseltseva, 2018).

5. Ejemplos de introducción del modelo agile en la empresa

En el apartado anterior se ha podido ver, de forma teórica, qué es la *metodología Agile*, qué beneficios genera en la empresa, y qué se debe hacer para implementarlo en la misma. A continuación, se plantearán algunos ejemplos de cómo grandes empresas, de diferentes sectores, han conseguido implementar con éxito esta metodología: **Orange SA** (sector tecnología y telecomunicaciones) y el **BBVA** (sector financiero).

5.1 Orange SA

Orange, compañía francesa de telecomunicaciones, es una de las líderes del mercado en este sector, reteniendo un 18,4% de cuota de mercado a finales del tercer trimestre de 2018, a la par con Vodafone (18,6%) y solamente superada por Movistar (CNMC, 2019). A pesar de ser una de las líderes, como se ha visto en anteriores apartados, una empresa con visión de futuro siempre debe renovarse y adaptarse a los cambios que puedan surgir en su sector. Uno de estos cambios son los cada vez más exigentes gustos del consumidor, sobre todo en este sector, en el que el trato con el cliente es fundamental. Para ello, Orange tomó la decisión de introducir la *metodología Agile,* concretamente en la unidad de negocio Digital, dedicada a digitalizar cada una de las acciones que un cliente pueda querer realizar con la empresa (Orange, 2018). La razón por la que decidieron implementar esta metodología en esta unidad es sencilla: son los que más rápido deben adaptarse a los cambios en las necesidades y gustos del cliente, pues cada vez demandan más la digitalización de las interacciones entre ellos y la empresa, ya sea a través de sus

smartphones, ordenador u otro dispositivo, y la *metodología Agile* puede proporcionarles dicha agilidad (Orange, 2018).

El principal cambio que llevaron a cabo fue la organización de los equipos: se crearon **equipos multifuncionales**, de entre 8 y 10 personas, en los que cada miembro, con capacidades similares, pero distintos perfiles, aportaba un valor diferente, creando un conjunto de personas capaces de hacer frente a cualquier tipo de cambio, de la forma más rápida, eficiente, y con más calidad posible. Mezclar personas con perfil de negocio con otras de perfil más técnico permite que se pueda abordar cualquier tipo de problema, teniendo siempre en mente qué es lo que quiere el cliente, y que lo quiere de la forma más sencilla posible (Orange, 2018).

Este cambio organizacional supuso también una nueva distribución de responsabilidades entre todos los integrantes de la unidad Digital. En la *metodología Agile*, se destruyen los complejos sistemas de jerarquías de las metodologías tradicionales, para alinear tanto a trabajadores como a managers, lo que supone que los trabajadores asumirán responsabilidades que antes no tenían, y que los managers pasan de tomar decisiones en solitario, a llegar a una solución final con el resto del equipo (Orange, 2018).

A pesar de que estos cambios pudieron conllevar algunas dificultades, como la implantación de una nueva y revolucionaria forma de trabajo y organización en una empresa, la asunción de responsabilidades por parte de gente que no está acostumbrado a ello, o el uso de nuevas herramientas y técnicas optimizadas para satisfacer al cliente (muy específico) trabajando con datos de más de 20 millones de clientes (muy genérico), el impacto de la implantación de la metodología ha sido positivo: la **comunicación** entre miembros del equipo se mejora gracias a la eliminación de las jerarquías, la **creatividad** del equipo aumenta, gracias a la mencionada multifuncionalidad, y, sobre todo, la **satisfacción del cliente** ha aumentado, pues se realizan entregas con el triple de frecuencia, y más adaptadas a sus gustos y necesidades, pues el equipo *agile* está constantemente en contacto con el cliente (Orange, 2018).

5.2 BBVA

El Banco Bilbao Vizcaya Argentaria (popularmente conocido como BBVA) es una de las entidades financieras más prestigiosas del mundo; a nivel nacional, se posiciona en el segundo puesto del ranking de entidades financieras con mayor valor de marca , cotizando en el Ibex 35, teniendo también presencia a nivel internacional en toda Europa, Estados Unidos, Hispanoamérica, China y Turquía (MarketingNews, 2019).

La aventura del BBVA con la *metodología Agile* comenzó en el año 2014, cuando decidieron implementarla en el desarrollo de aquellos proyectos (en España y México) en los que se estaba desarrollando un producto para el cliente, con el único objetivo de reducir los tiempos de entrega de dichos productos al cliente (Fernández Espinosa, 2019). Tal fue el éxito de dicha implantación (además de reducir los tiempos de entrega, tuvo otros beneficios como que los miembros de los equipos podían ver reflejado el fruto de su esfuerzo en la satisfacción directa del cliente), que decidieron expandir la implantación de la metodología a todas las filiales del banco. Un ejemplo de dicho éxito fue la implementación de la aplicación del BBVA para dispositivos móviles, que obtuvo el premio a "mejor app del mundo" (La Vanguardia, 2018; Fernández Espinosa, 2019).

La siguiente decisión que tomó el BBVA, vistos los beneficios de la *metodología Agile* en la práctica, fue la de implantarla en todas las áreas centrales de la entidad, lo que suponía establecer dicha metodología como la predominante en la forma de trabajar de la empresa, llegando a unos 33000 empleados a los que hubo que enseñar a trabajar siguiendo los estándares *Agile* (Fernández Espinosa, 2019). Para ello, se contrató a unos 260 *Agile Coaches* en las diferentes geografías de la entidad, para transmitir la metodología a cada uno de los empleados del BBVA, creando hasta una escuela interna para formar a dichos entrenadores (Fernández Espinosa, 2019).

Con todas estas modificaciones, se lograron los dos objetivos que se buscaban implementado la *metodología Agile*: se consiguieron alinear a lideres de equipo y miembros de equipo, siendo más fácil transmitirles la estrategia que quieren que sigan, con una supervisión más directa, pudiendo incluso obtener *feedback* de ellos, y también se consiguió transformar la forma de trabajo, haciéndola más dinámica y flexible a cambios, realizando reuniones trimestrales para valorar el desarrollo del producto de cada equipo, mostrando resultados reales (esto es, el producto en sí), pudiendo llegar a redistribuir recursos para los equipos según estos resultados, en caso de que fuese necesario.

B. DATOS

Para realizar el análisis que se desarrolla en los siguientes apartados, se han utilizado datos sobre el ***Beneficio antes de impuestos (BAI), número de empleados e inversión en i+d+i*** de diez de las entidades financieras más importantes de España y Europa, desde el año 2015 hasta el año 2018, ambos incluidos. A la hora de tratar los datos para el análisis, tanto el *BAI* como la *inversión en i+d+i* se han utilizado en *millones de euros (*con la abreviación *MM€*), para que dicho análisis sea más fácil de realizar e interpretar, puesto que las cifras en euros de ambos datos son demasiado altas.

Para obtener el *BAI* y el *número de empleados*, se ha utilizado la base de datos ***Osiris***, de la conocida empresa de análisis y recolección de información *Bureau Van Dijk*, que dispone de datos financieros de todas las empresas del mundo que cotizan en bolsa (Bureau Van Dijk, 2019). Se pueden ver los datos recogidos de esta base de datos en el Anexo I.

El dato sobre la *inversión en i+d+i* de cada una de las empresas incluidas en el análisis no se puede encontrar en *Osiris*, por lo que se ha recurrido a cada una de las memorias anuales de dichas empresas para acceder a este dato (la información sobre estos documentos se puede encontrar en la bibliografía). Dada la imposibilidad de encontrar la cifra exacta de la inversión que cada empresa ha realizado en la incorporación de la *metodología Agile* en su forma de trabajo, se ha utilizado el dato de *inversión en i+d+i* en su defecto, pues es el dato más específico que engloba, entre otras inversiones, la realizada para introducir la *metodología Agile*, y que se puede encontrar en los informes anuales de las empresas que se han estudiado.

C. METODOLOGÍA

Para el análisis de la eficiencia de las empresas que han incorporado la *metodología Agile* en su forma de trabajo que se ha llevado a cabo en este trabajo, se ha realizado un ***Análisis Envolvente de Datos***, comúnmente conocido como ***DEA*** (del inglés *Data Envelopment Analysis*). Antes de comenzar con la explicación del modelo, se realizará una pequeña introducción de este.

1. Análisis Envolvente de Datos (DEA)

El conocido como *análisis DEA* es un método cuya principal función es definir la **eficiencia** (entendiendo por eficiencia la "capacidad de una unidad productiva de obtener un *output* con el menor número de recursos o *inputs* posibles") de un grupo de unidades productivas, llamadas *Unidades de Toma de Decisiones* (Schuschny, 2007), a partir de ahora ***DMU's*** (del inglés *Decision Making Units*), que realizan actividades similares (Eken & Kale, 2011), utilizando la denominada ***frontera de eficiencia***, en la que se situarán aquellas *DMU* que, obteniendo una eficiencia del 100%, sirven como referencia al resto (Ramírez & Alfaro, 2013).

Este método fue desarrollado en el año 1978 por Charnes, Rhodes y Cooper, apoyándose en el trabajo iniciado en 1957 por M.J. Farrel sobre las medidas de la eficiencia de la productividad (Ramírez & Alfaro, 2013). Este método se basa, básicamente, en la generación de un coeficiente de eficiencia calculado como el cociente entre la suma ponderada de los ***outputs*** generados por la *DMU*, entre la suma ponderada de sus ***inputs*** (García Fariñas, 2009).

A la hora de intentar definir la eficiencia de una empresa, este método puede servir como perfecto complemento a un análisis cualitativo simple, con el que se puede obtener otro tipo de información importante para la toma de decisiones, pero que tiene un factor de subjetividad que puede ser contrarrestado con los **datos cuantitativos** a obtener del *análisis DEA* (Ramírez & Alfaro, 2013). También es una **ventaja** de este método el hecho de que se pueden tener en cuenta todos y cada uno de los ***inputs***, independientemente de su unidad de medida (por ejemplo, número de horas trabajadas, ingreso en euros…), así como tampoco importa lo diferentes que sean los ***outputs***. Cuantos más de ambos incluyamos en el modelo (siempre que sean homogéneos y tengan coherencia con el caso de estudio), más preciso será este (Schuschny, 2007).

Sin embargo, este método también tiene **limitaciones**, como, por ejemplo, el hecho de que es posible que estemos obviando variables (tanto *inputs* como *outputs*) en el modelo, que fuesen relevantes, y por lo tanto la precisión de la medida de la eficiencia disminuya (Schuschny, 2007). También hay que tener cuidado con la homogeneidad de las *DMU* estudiadas, pues una inclusión en el método de *DMU's* de diferente naturaleza (por ejemplo, una universidad con un hospital) puede llevar también a resultados incorrectos (Ramírez & Alfaro, 2013).

A pesar de sus limitaciones, el *análisis DEA* se utiliza con bastante frecuencia, y, si se tienen en cuenta sus limitaciones, puede llegar a ser un poderoso aliado a la hora de intentar establecer un coeficiente de eficiencia en entidades tanto privadas como públicas (García Fariñas, 2009).

2. Representación matemática del modelo

En este método, se analizará la eficiencia de todas y cada una de las *DMU* que incluyamos en el modelo. Se supondrá un número *N* de *DMU's*, y un proceso de producción en el que se utilizan *n* inputs, representados en el vector $x_f = (x_{f0}, x_{f1}, \dots, x_{fn})$, y *m* outputs, representados en el vector $y_f = (y_{f0}, y_{f1}, \dots, y_{fm})$, donde *f* representa la *DMU* que estamos analizando pues, como se verá más adelante, hay que repetir el problema por cada una de las *DMU* que hayamos incluido en el modelo (Schuschny, 2007).

El problema a resolver sería el siguiente: encontrar los ponderadores $u_i = (u_0, u_1, \dots, u_n)$ para los inputs y $v_j = (v_0, v_1, \dots, v_n)$ para los outputs, ambos con valores positivos, que maximicen la productividad ponderada por dichos valores (a la que llamaremos *P*), siendo esta productividad siempre igual o menor a 1, e igual al cociente entre outputs e inputs (Schuschny, 2007).

Matemáticamente, se puede expresar como un problema de optimización como el siguiente (se representará para *f = 0*, que sería la primera de las *DMU* de la lista):

$$\max_{u,v} P_0 = \frac{\sum_{j=1}^{m} v_j y_{j0}}{\sum_{i=1}^{n} u_i x_{i0}}$$

$$s.a.:$$

$$\frac{\sum_{j=1}^{m} v_j y_{jr}}{\sum_{i=1}^{n} u_i x_{ir}} \leq 1 \quad r = 1,2 \dots, N$$

$$u_i, v_j \geq 0$$

Esta sería la modelización del problema, aunque habrá que retocar algunas cosas para que el cálculo de la eficiencia sea óptimo. Lo primero será eliminar el denominador de la función objetivo para simplificar el cálculo de la productividad ponderada y evitar futuros problemas en dicho cálculo, esto es, transformar el problema, pasando de ser un Problema Fraccional a un Problema Lineal. Para ello, simplemente hay que añadir una restricción al problema, en la que forcemos al denominador del producto a ser igual a 1 (Schuschny, 2007). Matemáticamente:

$$\sum_{i=1}^{n} u_i x_{i0} = 1$$

La otra modificación que se debe realizar para transformar el problema es en los pesos, que quedarán configurados de la siguiente forma (Schuschny, 2007):

$$\alpha_i = t u_i$$

$$\beta_j = t v_j$$

$$donde \quad t = \frac{1}{\sum_{i=1}^{n} u_i x_{io}}$$

Así, y cambiando la nomenclatura de los pesos (siendo ahora α el peso asociado a los inputs, y β el asociado a los outputs), se transforma el problema en uno lineal, que matemáticamente quedaría expresado de la siguiente forma (Schuschny, 2007):

$$\max_{\alpha,\beta} P_0 = \sum_{j=1}^{m} \beta_j y_{j0}$$

$$s.a.:$$

$$\sum_{i=1}^{n} \alpha_i x_{i0} = 1$$

$$\sum_{j=1}^{m} \beta_j y_{jr} - \sum_{i=1}^{n} \alpha_i x_{jr} \leq 0 \qquad r = 1,2 \dots, N$$

$$\alpha_i, \beta_j \geq 0$$

Todo problema lineal tiene un problema dual asociado con propiedades importantes en la toma de decisiones con respecto al problema inicial, llamado también problema primal (Salazar López, 2016). Por ello, la ultima modificación que se realizará en el problema plantado al principio, será obtener dicho problema dual, que quedaría, matemáticamente, expresado de la siguiente forma (Schuschny, 2007):

$$\min_{\theta,\lambda} \theta$$

$$s.a.:$$

$$Y\lambda \geq y_0$$

$$\theta x_0 - X\lambda \geq 0$$

$$\lambda \geq 0$$

Donde:

-*X* e *Y* son las matrices de los *outputs* e *inputs*, respectivamente, con tantas filas como *DMU* haya, y tantas columnas como *inputs/outputs* haya.

-λ es el vector de pesos, $\lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \dots \\ \lambda_n \end{pmatrix}$, donde *n* es el número de DMU en el modelo.

-θ es la eficiencia de la *DMU* en estudio.

Este sería el modelo matemático de un *análisis DEA*, en el que se resolverá el problema que se planteó anteriormente, buscando una solución para θ para cada una de las *DMU*, una solución que será siempre, como máximo, 1, en cuyo caso se interpreta que dicha unidad es eficiente. En el caso en que dicho valor sea menor que 1, se interpreta que dicha unidad es ineficiente. El problema también obtendrá los valores de λ óptimos para obtener dicha eficiencia θ (Schuschny, 2007).

3. Descripción del caso de estudio

Una vez descrito el modelo que se va a utilizar, se explicarán todas las partes que conformarán al mismo. En el análisis realizado en este trabajo, se han incluido hasta diez de las entidades financieras más importantes del panorama europeo, de las que se medirá su eficiencia, utilizando el *análisis DEA*, en términos de la rentabilidad que obtienen cada una sobre la inversión en *i+d+i* que cada una ha realizado en los diferentes años del estudio, y en términos del número de empleados que cada una tuvo en activo en dichos años. Por lo tanto, los *inputs* que se utilizarán en este estudio serán la *inversión en i+d+i* y el *número de empleados*, mientras que el *output* será el *Beneficio Antes de Impuestos*.

Dado que el *output* es el resultado que se obtiene del proceso de producción, se ha escogido el beneficio que cada empresa ha obtenido en cada ejercicio (antes de impuestos), seleccionando como *inputs* aquellos recursos que se han utilizado para obtenerlo, y que tienen relación directa con el objeto de estudio, que no es otro que la utilización de la *metodología Agile*.

Como ya se ha explicado en la sección de *Datos* del trabajo, se ha escogido *inversión en i+d+i* como *input* ante la imposibilidad de obtener un dato más específico sobre la inversión en la implantación de la *metodología Agile* en cada empresa. El *numero de empleados*, sin embargo, si que es un claro representante de la utilización de la *metodología Agile,* pues, como se ha visto anteriormente en el trabajo, es una característica clave en esta forma de trabajo, la utilización de equipos más pequeños, autosuficientes, multidisciplinares, y con menos jerarquía que los tradicionales, por lo que sería un claro indicativo de la introducción de dicha metodología cierta reducción en la plantilla de empleados de cada empresa.

El análisis se realizará hasta para cuatro años distintos, desde 2015 hasta 2018, dado que la *metodología Agile* empezó a introducirse y consolidarse durante estos años, de forma que se puedan apreciar ciertos cambios en la eficiencia de las empresas gracias a la introducción de esta metodología.

El análisis se realizará en un archivo de Excel, en el que se crearán hasta diez hojas de cálculo diferentes, correspondientes a cada una de las empresas a analizar, dentro de las cuales encontraremos cuatro tablas, correspondientes a los cuatro años en los que se realizará el estudio, en las que se recogerán los datos de ese año de cada una de las empresas, además de los elementos necesarios para realizar el *análisis DEA* (ya

explicados en el apartado anterior), que no son otros que el vector de pesos λ y la eficiencia θ (los cuales rellenaremos con un "1", pues nos da igual su valor, ya que será calculado por el programa), y las restricciones correspondientes al análisis de esa empresa. Una vez configuradas las tablas de los diferentes años de estudio de cada una de las empresas a analizar, se ha procedido a realizar el análisis, con la herramienta Solver de Excel, en la que se establece la función objetivo del problema (la eficiencia), las variables (la eficiencia y los pesos), y cada una de las restricciones (en nuestro caso, cuatro).

En las tablas 11 y 12 del Anexo II se puede ver un ejemplo de estas tablas, correspondiente a la empresa CAIXABANK en el año 2018, antes y después de haber realizado el análisis.

D. RESULTADOS

Tras la ejecución del análisis descrito en el apartado anterior, se han recogido todos los resultados de la eficiencia de cada *DMU* en cada uno de los años contemplados en el análisis, en dos tablas en la que se incluyen cada uno de los datos de *inputs* y *outputs* en cada uno de dichos años, junto al resultado de la eficiencia para los mismos, con el fin de poder realizar un análisis de los resultados de forma más sencilla. Estas tablas se pueden encontrar en el Anexo III (tablas 13 y 14).

Visualizando estas tablas, lo primero que llama la atención es el considerable aumento de la eficiencia de la mayoría de las *DMU* estudiadas, desde el 2015 hasta el 2018. Todas ellas, menos *Societe Generale*, han conseguido incrementar su eficiencia. Algunas como *Lloyds Bank, HSBC* y *Caixabank* han logrado mantener el 100% de eficiencia que consiguieron en 2015, mientras que otras como *ING* y *BNP Paribas* han conseguido situarse en la *frontera de eficiencia,* logrando en 2018 un 100% de eficiencia, junto a las tres *DMU's* mencionadas anteriormente.

Observando los datos de ambas tablas, llama la atención también como, en general, tanto el *BAI* como la *inversión en i+d+i*, van aumentando con el paso de los años, lo que hace pensar que haya una relación directa y mutua entre ellas, pues cuanto más beneficio obtenga una empresa, más dinero tendrá para invertir y, según los resultados arrojados por el análisis, parece que el beneficio también puede incrementar en función del gasto en *i+d+i* que realice una empresa. Dado que una empresa realiza

inversiones en innovación y desarrollo para adaptarse a los nuevos tiempos y tecnologías, y obtener algún tipo de ventaja competitiva con respecto a sus competidores, es normal que un incremento en dicha inversión tenga un efecto positivo sobre el beneficio.

Otro dato que llama la atención es el descenso del número de empleados con el paso de los años en la algunas de las entidades analizadas, siendo dos de estas empresas (concretamente *HSBC* y *Lloyds Bank*) de las cinco que se sitúan en la frontera de eficiencia en 2018 (las otras tres con un 100% de eficiencia tampoco han incrementado el número de empleados, se ha mantenido más o menos constante). Esto puede tener su explicación en el cambio de metodologías de trabajo y estructuración de las empresas, concretamente en la transformación de estas empresas hacia la *metodología Agile*, en la que, como ya se ha visto, se utilizan grupos más pequeños y sin jerarquías.

Representando los resultados obtenidos sobre la eficiencia de las entidades estudiadas, se ha podido definir la situación de cada una de forma gráfica, consiguiendo también representar la conocida *frontera de eficiencia*, en la que se sitúan aquellas *DMU* que obtuvieron una eficiencia del 100%, y que sirven de referencia para aquellas entidades ineficientes, que para alcanzar la eficiencia deben adaptar sus *inputs* y *outputs* de forma que se vayan acercando cada vez más a dicha frontera. En la figura 62 se puede observar dicha gráfica, en la que se ha utilizado como medida del eje vertical, el cociente entre el *BAI* y el *número de empleados,* y en el eje horizontal, el cociente entre el *BAI* y la *inversión en i+d+i*.

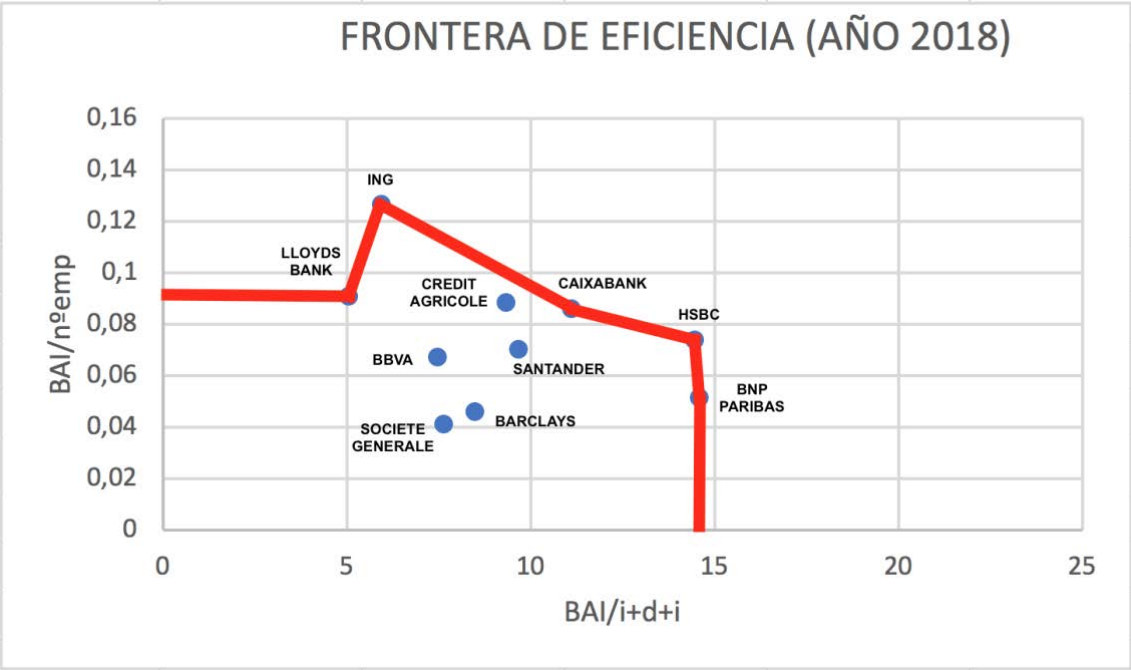


Figura 62. Representación de la frontera de eficiencia en el año 2018 (Elaboración propia)

Por último, y para poder ver la evolución a lo largo de estos años de la eficiencia de cada una de las *DMU* analizadas, se han realizado dos gráficos de frecuencias, en los que se recogen, por intervalos, que cantidad de *DMU* han obtenido un porcentaje de eficiencia u otro, tanto en el año 2015, como en el 2018. Se pueden ver dichos gráficos en las figuras 63 y 64.

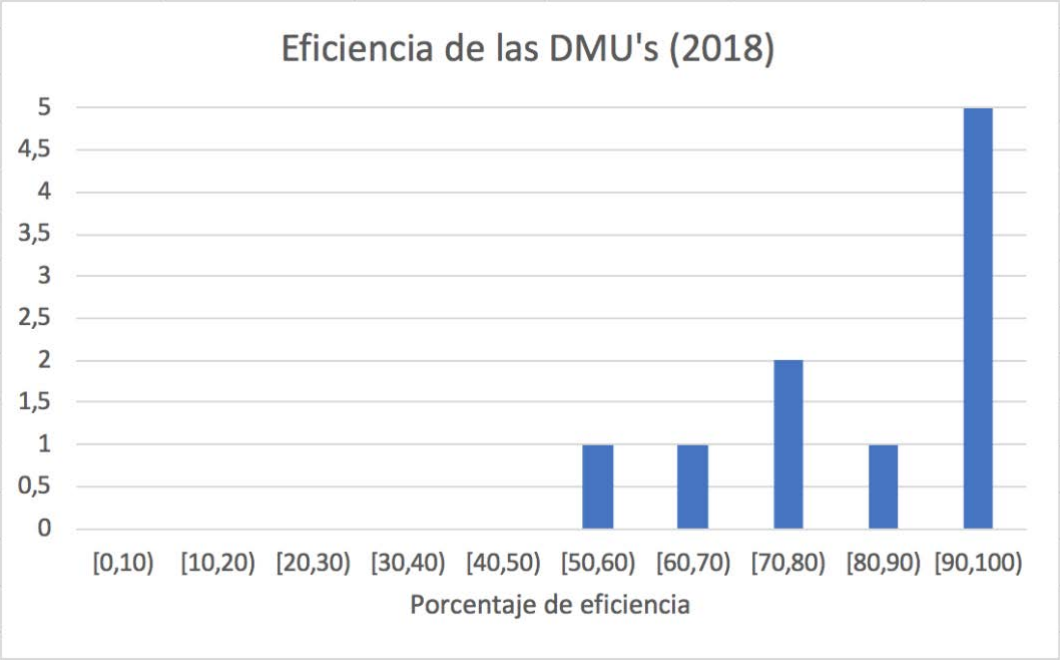


Figura 63. Gráfico de frecuencias sobre la eficiencia de las DMU's en 2018 (Elaboración propia)

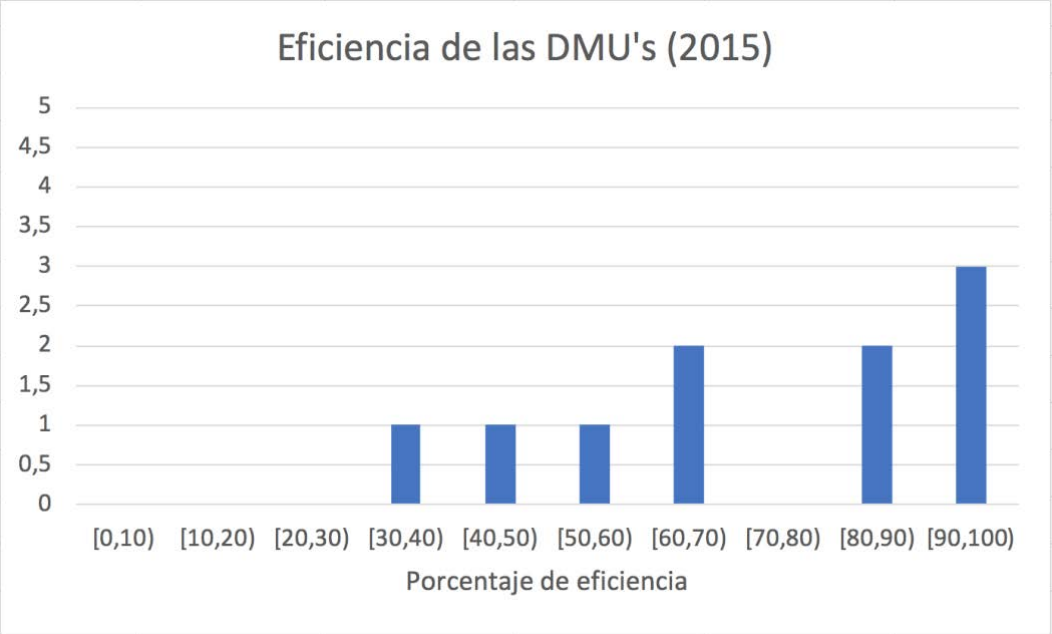


Figura 64. Gráfico de frecuencias sobre la eficiencia de las DMU's en 2015 (Elaboración propia)

IV. CONCLUSIONES

Una vez realizado el estudio sobre el ***Modelo Ágil*** de despliegue de aplicaciones y todos sus componentes, con respecto a su aplicación en la rama del software, se pueden extraer las siguientes conclusiones sobre el mismo:

1. Es **económico**, pues todo tipo de actividad que se desarrolle en un proyecto de software adaptado al *Modelo Ágil* es **reutilizable**, dado que los patrones de despliegue de una aplicación pueden ser similares, y también **escalables**, pues también pueden ser usados en entornos diferentes. Además, también hay ahorro en la reorganización de los equipos, más pequeños, compactos y eficientes, en la optimización del proceso de prueba del código, y, en definitiva, en cualquier cambio o imprevisto durante el desarrollo del proyecto, pues la adaptabilidad de este modelo permite realizar cambios de rumbo en el proyecto sin que la economía del mismo se resienta.
2. Es **rápido**, la automatización de procesos hace que el desarrollo del proyecto sea fluido, que cualquier cambio pueda ser introducido con un solo *click* de ratón, que una aplicación se pase las pruebas y se despliegue en cuestión de minutos.
3. Es **fiable** y **seguro**, antes del despliegue, el código pasa por un proceso de pruebas y compilación, en el que se filtrará cualquier error que pueda haberse introducido en el *mainline*. Los errores humanos también se verán reducidos gracias a la automatización del proceso.
4. Es **transparente**, pues cualquier persona relacionada con el proyecto puede acceder, en cualquier momento, a cualquier tipo de información relativa al proyecto. Esto también le otorga al *Modelo Ágil* la característica de ser **auditable**, pues, al igual que cualquier integrante del proyecto puede acceder a cualquier tipo de información sobre el mismo, también podrá hacerlo cualquier persona o empresa que tenga que desempeñar la tarea de auditar el proyecto.

En cuanto al **entorno de desarrollo de integración continua** que se ha configurado y utilizado en este trabajo, son también varias las conclusiones que se pueden extraer:

1. En un proyecto en el que se trabaja en equipo, un entorno de integración continua permite una perfecta comunicación entre todos los integrantes de este, evitando los frecuentes problemas de integración de código, gracias a las herramientas de integración y control de versiones que se han utilizado en este trabajo.
2. En un entorno de integración continua bien montado, todos los procesos están tan automatizados que lo único de lo que tiene que preocuparse el desarrollador es de implementar su código. Una vez terminado, con solo pulsar un botón, el resto de las actividades del flujo de trabajo se realizan por si solas.
3. Si bien un entorno de integración continua es fácil de usar, montarlo y configurarlo no es una tarea tan sencilla, la gran diferencia entre las diferentes plataformas y servidores donde se pueden implementar hace que no sea tan fácil implementar un entorno si la forma en la que el usuario está intentando hacerlo no está documentada (como es el caso de este trabajo, en el que se ha configurado un entorno de integración continua con un servidor de *Google Cloud*, para el cual no existía ningún tipo de documentación). La parte positiva es que este trabajo podrá servir en un futuro como guía para todo aquel que desee implementar este tipo de entorno con la herramienta de *Google*.

En lo que a la aplicación de la *Metodología Agile* en las empresas se refiere, y al análisis sobre el efecto de la aplicación de dicha metodología en la eficiencia de dichas empresas, cabe destacar, tras el análisis cualitativo (estudio previo acerca de cómo se aplica esta metodología en las empresas, y los ejemplos que se han visto en el trabajo) y el análisis cuantitativo (*análisis DEA*), el impacto que la implantación de esta forma de trabajo parece tener sobre las empresas, ya sea sobre la **eficiencia**, que a lo largo de los años ha ido mejorando considerablemente, como sobre los **beneficios**, o incluso sobre la **satisfacción** de los empleados, tanto subordinados como jefes, que han visto como la estructura jerarquizada a la que estaban acostumbrados, a dado paso a una estructura más

horizontal, en la que la comunicación entre todos los integrantes del proyecto mejora, eliminando la presión que caracterizaba a dicha estructura.

Este cambio en la estructura de los proyectos también acarrea una reubicación y optimización de los recursos humanos de la empresa, pues con menos gente se pueden conseguir los mismos *outputs,* trabajando de una forma diferente. Así se ve reflejado en los datos recogidos sobre las empresas analizadas, y en los resultados del análisis, en los que podemos ver que las empresas más eficientes, han mantenido o disminuido el número de empleados a lo largo de los años.

El efecto positivo que la utilización de la *metodología Agile* parece tener sobre las empresas llevadas a estudio, puede tener su origen en la naturaleza de dichas empresas. Al ser todas ellas entidades financieras, en las que la relación con el cliente es fundamental, esta metodología puede ser la más apropiada, incluyendo al cliente en el desarrollo de proyecto, de forma que el producto esté siempre adaptado a sus necesidades, que pueden cambiar de un día para otro. El constante despliegue del producto (como se ha visto en el *Modelo Ágil* de despliegue de aplicaciones), que a su vez siempre está adaptado a las necesidades y gustos del cliente, hace que la empresa pueda obtener rentabilidad de su producto mientras lo está desarrollando, y no solo al finalizar el proyecto, razón por la cual el beneficio de las empresas va aumentando año a año (como se puede ver en las tablas 11 y 12 del Anexo II).

Respecto a las limitaciones que se han encontrado a la hora de realizar el trabajo, caben destacar, por la parte del desarrollo del entorno de integración continua, la poca o nula documentación existente para ayudar a configurar un entorno de estas características utilizando servidores online (existe bastante documentación para hacerlo de forma local, en nuestros ordenadores), si bien, extrayendo información de varias fuentes, ha sido posible implementarlo, y documentarlo de forma que, a partir de ahora, cualquiera que necesite o quiera instalar un entorno de la forma en que se ha hecho en este trabajo, tenga un apoyo para convertir esta tediosa tarea en sencilla. Otra de las limitaciones que se ha encontrado ha sido la imposibilidad de integrar la herramienta de pruebas de *Javascript*, *QUnit*, con *Jenkins*, de forma que la ejecución de pruebas se realizara de forma automática, por lo que se ha establecido en este trabajo una forma alternativa para realizar las pruebas que, si bien no es automática, no debería suponer problemas a la hora de no filtrar posibles errores, siempre que el desarrollador las siga al pie de la letra.

En la parte del *análisis DEA* que se ha realizado para el estudio del efecto que la introducción de la *metodología Agile* puede tener sobre la eficiencia de las empresas, las principales limitaciones han sido, por una parte, el hecho de tener que asumir que la *inversión en i+d+i* equivale al gasto de las empresas en introducir la *metodología Agile*, al no poder contar con este dato. Asumiendo esto, podemos estar contemplando otros gastos en tecnología que no tengan que ver con la introducción de esta metodología. Otro de los factores que podrían haber limitado el análisis, sería el hecho de no haber asumido ciertas variables en el estudio que pudieran haber sido relevantes, hecho que podría reducir la precisión de este análisis.

Por último, respecto a los pasos a seguir después de la realización de este trabajo, un posible trabajo futuro sería la configuración de un entorno más complejo y completo, que incorpore herramientas como, por ejemplo, un sistema de control de bases de datos como *MySQL*, o una herramienta de evaluación de código como *SonarCube*, desarrollando un proyecto en otro lenguaje diferente a *Javascript*, para complementar este trabajo, teniendo en cuenta cualquier variante que pueda darse por la utilización de otras herramientas o lenguajes de programación. El análisis del efecto de la *metodología Agile* en la empresa, por su parte, se podría complementar con un estudio sobre la influencia de esta metodología en empresas de otros sectores diferentes al financiero, de forma que se pueda ver si, efectivamente, su efecto es positivo en cualquier tipo empresa, o si por el contrario solo encaja en el sector financiero.

V. BIBLIOGRAFÍA

Agile Alliance. (2019). *Agile Alliance*. Obtenido de https://www.agilealliance.org/glossary/continuous-deployment/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'continuous*20deployment))~searchTerm~'~sor

Alfaro Giménez, J., & Pina Massachs, M. (2018). *Empresa y administración.* McGraw-Hill Education. Obtenido de MHEducation: https://www.mheducation.es/bcv/guide/capitulo/8448614224.pdf

Ambler, S. W. (2003). *AgileData*. Obtenido de http://agiledata.org/essays/tdd.html

Anderson, D. J., & Carmichael, A. (2016). *Essential Kanban Condensed.* Washington: LeanKanban University Press.

Armengol, L. (24 de Octubre de 2018). *InnovaSpain*. Obtenido de https://www.innovaspain.com/grupo-santander-eleva-inversion-ciberseguridad-400-millones-euros/

ARSYS. (26 de Enero de 2017). *Blog de Arsys*. Obtenido de https://www.arsys.es/blog/programacion/tomcat-servidores-cloud/

Asociación de la Industria Navarra. (2008). *Guía práctica: La gestión de la innovación en 8 pasos.* Pamplona: Agencia Navarra de Innovación.

Barclays PLC. (2016). *Annual Report 2015.*

Barclays PLC. (2017). *Annual Report 2016.*

Barclays PLC. (2018). *Annual Report 2017.*

Barclays PLC. (2019). *Annual Report 2018.*

BBVA. (2016). *BBVA en 2015.*

BBVA. (2017). *BBVA en 2016.*

BBVA. (2018). *BBVA en 2017.*

BBVA. (2019). *BBVA en 2018.*

Beck, K. (2001). Obtenido de Agile Manifesto: https://agilemanifesto.org/iso/es/manifesto.html

Beck, K., & Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.

Beck, K., & Gamma, E. (2004). *Extreme Programming Explained: Embrace Change* (2ª edición ed.). Addison-Wesley.

Benítez, C. (2011). *EtnasSoft*. Obtenido de http://www.etnassoft.com/2011/02/01/qunit-testeando-nuestras-aplicaciones-javascript/

Blé, C. (2010). *Diseño Ágil con TDD.* Safe Creative.

BNP Paribas. (2016). *Registration DOcument and Annual Financial Report. 2015.*

BNP Paribas. (2017). *Registration Document and Annual Financial Report. 2016.*

BNP Paribas. (2018). *Registration Document and Annual Financial Report. 2017.*

BNP Paribas. (2019). *Registration Document and Annual Financial Report. 2018.*

Bureau Van Dijk. (2019). *Osiris*. Obtenido de https://osiris.bvdinfo.com/

Bustos, G. (16 de Julio de 2019). *Hostinger*. Obtenido de https://www.hostinger.es/tutoriales/que-es-apache/

Cabrera, D. (11 de Marzo de 2019). *VozPopuli*. Obtenido de https://www.vozpopuli.com/economia-y-finanzas/Santander-Boadilla-centro-ciberseguridad-empleados_0_1225677741.html

Christensen, C. M. (1997). *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Boston: Harvard Business Review Press.

CNMC. (31 de Enero de 2019). *Comisión Nacional de los Mercados y la Competencia*. Obtenido de https://www.cnmc.es/node/373272

Credit Agricole SA. (2016). *Annual Financial Report 2015. Registration Document.*

Credit Agricole SA. (2017). *Annual Financial Report. Registration Document 2016.*

Credit Agricole SA. (2018). *Annual FInancial Report. Registration Document 2017.*

Credit Agricole SA. (2019). *Annual FInancial Report. Registration Document 2018.*

Eclipse Foundation Inc. (2019). *Eclipse.org*. Obtenido de https://www.eclipse.org/downloads/packages/

Eken, M., & Kale, S. (2011). Measuring bank branch performance using data envelopment analysis (DEA): The case of Turkish bank branche.

El Economista. (2019). *El Economista: Ranking Nacional de Empresas por Facturación.* Obtenido de https://ranking-empresas.eleconomista.es/ranking_empresas_nacional.html

Escorsa, P., & Valls, J. (2003). *Tecnología e innovación en la empresa.* Barcelona: Ediciones UPC.

European Commission. (2018). *Economics of industrial Research and Innovation: Scoreboard 2018*. Obtenido de http://iri.jrc.ec.europa.eu/scoreboard18.html

Fernández Espinosa, L. (5 de Abril de 2019). *BBVA.com*. Obtenido de https://www.bbva.com/es/agile-es-un-viaje-de-transformacion-y-aprendizaje-continuo-para-bbva/

Fernández Sánchez, E., & Vázquez Ordás, C. J. (1996). El proceso de innovación tecnológica en la empresa. *Investigaciones europeas de dirección y economía de la empresa, II*(1), 29-45.

Finley, K. (14 de Julio de 2012). *TechCrunch*. Obtenido de https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=wW4Y3fHPI5UGO4iWqc3Vfw

Fowler, M. (Mayo de 2006). Obtenido de https://martinfowler.com/articles/continuousIntegration.html

Galbraith, J. K. (1980). *El nuevo estado industrial.* Barcelona: Ariel.

Galiana, P. (1 de Marzo de 2019). *Marketing4ECommerce*. Obtenido de https://marketing4ecommerce.net/la-importancia-de-implementar-la-metodologia-agile-en-una-empresa/

Gallardo, D. (26 de Noviembre de 2012). *IBM*. Obtenido de https://www.ibm.com/developerworks/ssa/library/os-ecov/index.html

García Fariñas, A. (2009). El análisis envolvente de datos, herramienta para la medición de la eficiencia en instituciones sanitarias, potencialidades y limitaciones.

García, A. (13 de Julio de 2018). *DoctorMetrics*. Obtenido de https://www.doctormetrics.com/google-cloud-platform/

Garzás, J. (29 de Noviembre de 2012). *JavierGarzas.com*. Obtenido de https://www.javiergarzas.com/2012/11/entrega-continua-continuous-delivery.html

Garzás, J. (2014). *JavierGarzas.com*. Obtenido de https://www.javiergarzas.com/2014/12/bdd-behavior-driven-development-1.html

Garzás, J. (9 de Mayo de 2014). *JavierGarzas.com.* Obtenido de https://www.javiergarzas.com/2014/05/jenkins.html

Gil, A. (3 de Abril de 2019). *Directivos Y Empresas*. Obtenido de https://www.directivosyempresas.com/que-es-la-metodologia-agile-y-porque-triunfa-en-las-grandes-empresas/

Giret, L. (26 de Octubre de 2018). *OnMSFT*. Obtenido de https://www.onmsft.com/news/its-official-github-is-now-owned-by-microsoft

Github Inc. (2019). *Github*. Obtenido de https://github.com/

Google. (2019a). *Google Cloud*. Obtenido de https://cloud.google.com

Google. (2019b). *Por qué elegir Google Cloud*. Obtenido de https://cloud.google.com/why-google-cloud/

Grupo Caixabank. (2016). *Informe corporativo integrado. 2015.*

Grupo Caixabank. (2017). *Informe corporativo integrado. 2016.*

Grupo Caixabank. (2018). *Informe corporativo integrado. 2017.*

Grupo Caixabank. (2019). *Informe de gestión consolidado del grupo Caixabank. 2018.*

Grupo Santander. (2017). *Informe de auditoría independiente, Cuentas anuales consolidadas e informe de gestión consolidado a 31 de diciembre de 2016.* Madrid.

Grupo Santander. (2018). *Informe de auditoría y cuentas anuales. 2017.* Madrid.

Grupo Santander. (2019). *Informe de gestión consolidado y cuentas anuales consolidadas. 2018.* Madrid.

Highsmith, J., & Cockburn, A. (Noviembre de 2001). Agile Software Development: The People Factor. *IEEE Computer, 34*(11), págs. 131-133.

Hito Master DAP. (2017). *MDAP Executive Master Project Management*. Obtenido de https://uv-mdap.com/programa-desarrollado/bloque-iv-metodologias-agiles/metodologias-agiles-vs-tradicionales/

HSBC Holdings PLC. (2016). *Annual Report and Accounts. 2015.* Londres.

HSBC Holdings PLC. (2017). *Annual Report and Accounts. 2016.* Londres.

HSBC Holdings PLC. (2018). *Annual Report and Accounts. 2017.* Londres.

HSBC Holdings PLC. (2019). *Annual Report and Accounts. 2018.* Londres.

Humble, J. (2017). *Continous Delivery*. Obtenido de https://continuousdelivery.com/

Humble, J., & Farley, D. (2011). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Indiana: Addison-Wesley.

IDG. (28 de Abril de 2018). *ComputerWorld*. Obtenido de https://www.computerworld.es/tendencias/el-70-de-las-grandes-empresas-espanolas-usan-agile-de-forma-regular

ING Groep N.V. (2016). *ING Group Annual Report 2015.* Amsterdam.

ING Groep N.V. (2017). *ING Group Annual Report. 2016.* Amsterdam.

ING Groep N.V. (2018). *ING Group Annual Report. 2017.* Amsterdam.

ING Groep N.V. (2019). *ING group Annual Report. 2018.* Amsterdam.

Integra IT. (2018). *Integra IT*. Obtenido de https://www.integrait.com.mx/blog/fintech-ecosistema-innovacion-financiera/

Jenkins. (2019). *Jenkins*. Obtenido de https://jenkins.io/

Kanbanize. (2019). *Kanbanize*. Obtenido de https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban/

La Vanguardia. (2 de Octubre de 2018). *La Vanguardia*. Obtenido de https://www.lavanguardia.com/economia/20181002/452141992615/bbva-app-mejor-mundo-banca-movil-forrester.html

Lloyds Banking Group. (2016). *Annual Report and Accounts. 2015.*

Lloyds Banking Group. (2017). *Annual Report and Accounts. 2016.*

Lloyds Banking Group. (2018). *Annual Report and Accounts. 2017.*

Lloyds Banking Group. (2019). *Annual Report and Accounts. 2018.*

Machuca, J. M. (17 de Mayo de 2018). *We Are Marketing*. Obtenido de https://www.wearemarketing.com/es/blog/que-es-la-metodologia-agile-y-que-beneficios-tiene-para-tu-empresa.html

MarketingNews. (4 de Febrero de 2019). *MarketingNews*. Obtenido de http://www.marketingnews.es/investigacion/noticia/1120211031605/once-bancos-espanoles-mas-valorados-del-mundo.1.html

Mestre Valdés, J. (4 de Septiembre de 2018). *Telefónica: Think Big / Empresas*. Obtenido de https://empresas.blogthinkbig.com/metodologia-agile-en-las-grandes-organizaciones-convertir-a-un-diplodocus-en-equipos-de-velociraptores/

Millán Alonso, S. (8 de Marzo de 2019). *El País Economía.* Obtenido de Cinco Días: https://cincodias.elpais.com/cincodias/2019/03/07/companias/1551982887_794044.html

Miró, A. (17 de Agosto de 2017). *Deusto Formación*. Obtenido de https://www.deustoformacion.com/blog/programacion-diseno-web/que-es-para-que-sirve-github

Montoya Suárez, O. (Agosto de 2004). Schumpeter, innovación y determinismo tecnológico. *Sciencia et Technica, X*(25).

Naranjo, A. F. (25 de Julio de 2016). Obtenido de El País.com.co: https://www.elpais.com.co/economia/por-que-es-tan-importante-innovar-conozca-algunos-casos-exitosos.html

North, D. (Marzo de 2006). *DanNorth.net*. Obtenido de https://dannorth.net/introducing-bdd/

Novoseltseva, E. (22 de Febrero de 2018). *Apiumhub*. Obtenido de https://apiumhub.com/es/tech-blog-barcelona/transformacion-agil-pasos-estadisticas/

OCDE. (2003). *Manual de Frascati: Medición de las actividades científicas y tecnológicas.* París: Fundación Española Ciencia y Tecnología (FECYT).

OCDE. (2006). *Manual de Oslo: Guía para la recogida e interpetación de datos sobre la innovación.*

Openbank. (2019). *Openbank.com*. Obtenido de https://www.openbank.es/quienes-somos

Orange. (1 de Agosto de 2018). *Orange.es*. Obtenido de http://blog.orange.es/empresas/susana-andujar-metodologia-agile-orange/

Pantaleo, G., & Rinaudo, L. (2015). *Ingeniería de Software.* Alfaomega Grupo Editor.

Patel, S. (14 de Octubre de 2014). *DZone*. Obtenido de https://dzone.com/articles/continuous-integration-how-0

Pressman, R. S. (2010). *Ingeniería del Software. Un enfoque práctico* (7ª edición ed.). McGraw-Hill.

Ramírez, P., & Alfaro, J. (2013). Evaluación de la Eficiencia de las Universidades pertenecientes al Consejo de Rectores de las Universidades Chilenas: Resultados de un Análisis Envolvente de Datos.

Salazar López, B. (2016). *IngenieriaIndustrialOnline*. Obtenido de https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/dualidad-en-programaci%C3%B3n-lineal/

Santander SA. (2018). *Santander SA: Informe Anual 2018*. Obtenido de https://www.santander.com/csgs/Satellite/CFWCSancomQP01/es_ES/pdf/INFORME_ANUAL_2018_ESP_Acc_JGA19.pdf

Schumpeter, J. A. (1996). *Teoría del desenvolvimiento económico*. Fondo de Cultura Económica USA.

Schuschny, A. R. (2007). El método DEA y su aplicación al estudio del sector energético y las emisiones de CO2 en América Latina y el Caribe.

Schwaber, K., & Sutherland, J. (2017). *Scrum.org*. Obtenido de https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100

Siles, F. (15 de Septiembre de 2015). *Xataka*. Obtenido de https://www.xataka.com/aplicaciones/un-dia-el-desarrollador-desperto-y-descubrio-que-github-se-habia-convertido-en-el-centro-de-la-programacion

Societe Generale. (2016). *Annual Financial Report. 2015.*

Societe Generale. (2017). *Annual Financial Report. 2016.*

Societe Generale. (2018). *Annual Financial Report. 2017.*

Societe Generale. (2019). *Integrated Report. 2018-2019.*

TechTarget. (Agosto de 2017). *SearchDataCenter*. Obtenido de https://searchdatacenter.techtarget.com/es/cronica/DevOps-efectivo-se-basa-en-automatizar-un-sistema-de-entrega-continua

Telefónica . (2016). *Telefónica Investigación y Desarrollo.* Obtenido de http://www.tid.es/es/innovacion-de-largo-plazo/como-trabajamos

Telefónica. (26 de Febrero de 2017). *Telefónica.* Obtenido de https://www.telefonica.com/web/sala-de-prensa/-/telefonica-presenta-aura-un-nuevo-modelo-de-relacion-con-sus-clientes-basado-en-inteligencia-cognitiva-pionero-en-el-sector

The Apache Software Foundation. (2019). *Apache Tomcat*. Obtenido de http://tomcat.apache.org/

The JQuery foundation. (2019). *QUnitJS*. Obtenido de https://qunitjs.com/

ThoughtWorks Inc. (2019). *ThoughtWorks*. Obtenido de https://www.thoughtworks.com/continuous-integration

Ubuntu. (2014). *UBUNTU Documentation*. Obtenido de https://help.ubuntu.com/lts/serverguide/httpd.html

Veloso, M. (27 de Septiembre de 2018). *ABC Economía.* Obtenido de https://www.abc.es/economia/abci-santander-refuerza-filial-digital-openbank-y-lanza-pulso-resto-bancos-y-fintech-201809270146_noticia.html

Wells, D. (2013). *Extreme Programming*. Obtenido de http://www.extremeprogramming.org/

WinLead. (Noviembre de 2018). *WinLead*. Obtenido de https://winlead.es/que-es-innovar-en-una-empresa/

ANEXO I. DATOS RECOGIDOS DE LA BASE DE DATOS OSIRIS

Tabla 1. Banco Santander SA

BANCO SANTANDER SA

39004 SANTANDER-CANTABRIA (ESPAÑA)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD ESA39000013
Estado Activa

Perfil financiero & empleados

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	1,459,271,000	1,444,305,000	1,339,125,000	1,340,260,000	1,266,296,000	1,115,638,000
Fondos propios	107,361,000	106,833,000	102,699,000	98,753,000	89,714,000	79,902,000
Ingresos operacionales	49,687,000	49,617,000	45,765,000	47,755,000	44,347,000	40,764,000
Resultado antes impuestos	14,201,000	12,091,000	10,768,000	9,547,000	10,679,000	7,652,000
Resultado neto	9,315,000	8,207,000	7,486,000	7,334,000	6,935,000	5,524,000
Capit. bursátil	64,507,909	88,409,987	72,313,829	65,792,418	88,040,563	73,735,234
Empleados	202,713	195,732	191,635	189,464	183,938	186,373

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 2. BBVA

BANCO BILBAO VIZCAYA ARGENTARIA SA - BBVA

48005 BILBAO (ESPAÑA)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD ESA48265169
Estado Activa

Perfil financiero & empleados

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	676,689,000	690,059,000	731,856,000	750,078,000	631,942,000	582,575,000
Fondos propios	52,874,000	53,323,000	55,428,000	55,439,000	51,609,000	44,850,000
Ingresos operacionales	25,856,000	27,491,000	26,756,000	25,472,000	23,088,000	23,060,000
Resultado antes impuestos	8,446,000	6,931,000	6,392,000	4,602,000	3,980,000	1,160,000
Resultado neto	6,151,000	4,762,000	4,693,000	3,328,000	3,082,000	2,981,000
Capit. bursátil	30,908,988	47,422,009	42,118,270	42,905,056	48,469,697	51,772,720
Empleados	125,627	131,856	134,792	137,968	108,770	109,305

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 3. ING Groep NV

ING GROEP NV

1000 AV AMSTERDAM (PAÍSES BAJOS)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD NL33231073
Estado Activa

Perfil financiero & empleados — Opciones — Columnas

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	887,030,000	846,216,000	845,081,000	841,769,000	992,856,000	1,081,317,000
Fondos propios	51,735,000	51,121,000	50,399,000	48,470,000	59,416,000	53,189,000
Ingresos operacionales	18,157,000	17,597,000	17,402,000	16,351,000	15,240,000	15,113,000
Resultado antes impuestos	6,838,000	7,268,000	5,903,000	6,172,000	3,707,000	4,167,000
Resultado neto	4,811,000	4,987,000	4,726,000	4,413,000	1,440,000	3,810,000
Capit. bursátil	36,620,706	59,549,053	51,854,827	48,178,359	41,784,515	38,786,943
Empleados	54,000	51,504	54,000	54,000	22,060	83,884

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 4. Caixabank SA

CAIXABANK, S.A.

VALENCIA (ESPAÑA)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD ESA08663619
Estado Activa

Perfil financiero & empleados — Opciones — Columnas

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	386,622,250	383,186,163	347,927,262	344,255,475	338,623,449	340,190,477
Fondos propios	24,058,445	24,683,281	23,555,562	25,204,550	25,232,568	24,333,772
Ingresos operacionales	8,774,097	8,466,184	7,860,035	8,651,993	7,364,667	7,473,486
Resultado antes impuestos	2,806,786	2,097,983	1,538,062	638,104	202,173	-712,851
Resultado neto	2,040,308	1,718,628	1,054,935	816,502	619,925	495,436
Capit. bursátil	18,925,270	23,261,812	18,781,716	18,718,305	24,922,923	18,774,406
Empleados	32,552	36,972	32,403	32,242	31,210	31,948

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 5. Credit Agricole SA

CREDIT AGRICOLE S.A.

92127 MONTROUGE (FRANCIA)
Empresa privada
El Global Ultimate Owner de esta participada es SAS RUE LA BOETIE

Número BvD FR784608416
Estado Activa

Perfil financiero & empleados — Opciones — Columnas

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	1,624,394,000	1,550,283,000	1,524,232,000	1,529,294,000	1,589,044,000	1,518,811,000
Fondos propios	65,516,000	64,706,000	63,937,000	59,435,000	56,161,000	47,883,000
Ingresos operacionales	19,736,000	18,715,000	17,120,000	17,476,000	15,982,000	16,076,000
Resultado antes impuestos	6,496,000	5,929,000	3,347,000	3,811,000	3,235,000	2,927,000
Resultado neto	5,027,000	4,217,000	3,955,000	3,971,000	2,760,000	2,885,000
Capit. bursátil	27,030,503	39,276,241	33,527,109	28,715,877	27,721,695	23,277,296
Empleados	73,346	73,707	70,830	71,495	72,567	78,532

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 6. Societe Generale SA

SOCIETE GENERALE SA

75009 PARIS (FRANCIA)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD FR552120222
Estado Activa

Perfil financiero & empleados

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	1,309,428,000	1,275,128,000	1,354,422,000	1,334,391,000	1,308,138,000	1,214,193,000
Fondos propios	65,809,000	64,037,000	65,706,000	62,675,000	58,874,000	53,970,000
Ingresos operacionales	25,218,000	24,125,000	25,937,000	24,983,000	23,733,000	22,658,000
Resultado antes impuestos	6,117,000	5,138,000	6,307,000	6,109,000	4,354,000	2,922,000
Resultado neto	4,556,000	3,430,000	4,338,000	4,395,000	2,978,000	2,394,000
Capit. bursátil	22,476,271	34,780,858	37,753,637	34,320,312	28,173,659	33,719,270
Empleados	149,022	147,125	145,672	145,703	148,322	147,682

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 7. Lloyds Banking Group PLC

LLOYDS BANKING GROUP PLC

EH1 1YZ EDINBURGH (REINO UNIDO)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD GBSC095000
Estado Activa

Perfil financiero & empleados

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Tipo de cambio: GBP/EUR	1.10878	1.12662	1.16706	1.36116	1.28556	1.19411
Total activo	884,357,749	914,935,280	954,415,439	1,098,035,269	1,099,020,826	1,005,896,130
Fondos propios	55,659,461	55,365,307	56,970,150	63,947,520	64,153,343	46,971,593
Ingresos operacionales	20,642,088	21,030,547	20,850,736	23,663,849	23,744,309	20,917,261
Resultado antes impuestos	6,608,307	5,942,901	4,946,010	2,237,755	2,265,158	495,557
Resultado neto	4,878,616	3,996,108	2,933,995	1,301,274	1,927,056	-957,678
Capit. bursátil	40,911,983	55,187,064	52,069,327	70,988,539	69,568,852	67,223,044
Empleados	72,626	75,944	80,418	85,703	95,088	97,869

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 8. Barclays PLC

BARCLAYS PLC

E14 5HP LONDON (REINO UNIDO)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD GB00048839
Estado Activa

Perfil financiero & empleados

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Tipo de cambio: GBP/EUR	1.10878	1.12662	1.16706	1.36116	1.28556	1.19411
Total activo	1,256,557,318	1,276,735,729	1,415,793,708	1,524,520,853	1,745,670,787	1,604,442,420
Fondos propios	70,716,643	74,374,705	83,287,406	89,651,755	84,793,022	76,362,273
Ingresos operacionales	23,435,096	23,744,566	25,075,502	29,414,770	27,933,952	33,651,272
Resultado antes impuestos	3,874,064	3,989,349	3,769,611	1,559,895	1,687,941	3,424,713
Resultado neto	2,630,017	-1,007,195	3,300,452	848,006	1,086,299	1,548,763
Capit. bursátil	28,593,456	39,036,886	44,236,758	50,070,821	51,644,936	52,326,169
Empleados	83,500	79,900	119,300	129,400	132,300	139,600

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 9. BNP Paribas SA

BNP PARIBAS SA

75009 PARIS (FRANCIA)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD FR662042449
Estado Activa

Perfil financiero & empleados — Opciones — Columnas

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Total activo	2,040,836,000	1,960,252,000	2,076,959,000	1,994,193,000	2,077,758,000	1,810,522,000
Fondos propios	105,726,000	107,209,000	105,220,000	100,077,000	93,641,000	90,955,000
Ingresos operacionales	42,516,000	43,248,000	43,190,000	42,957,000	39,702,000	37,400,000
Resultado antes impuestos	10,208,000	11,310,000	11,210,000	10,379,000	3,149,000	8,101,000
Resultado neto	8,005,000	8,207,000	8,115,000	7,044,000	507,000	5,421,000
Capit. bursátil	49,335,796	77,741,497	75,506,010	65,088,307	61,368,933	70,498,819
Empleados	197,162	196,128	192,419	189,077	187,903	n.d.

Fuente: Base de datos Osiris, Bureau Van Dijk.

Tabla 10. HSBC Holdings PLC

HSBC HOLDINGS PLC

E14 5HQ LONDON (REINO UNIDO)
Empresa privada
This company is the Global Ultimate Owner of the Corporate Group

Número BvD GB00617987
Estado Activa

Perfil financiero & empleados — Opciones — Columnas

Datos consolidados Pinchar aquí para otras cuentas	Cons 31/12/2018 mil EUR	Cons 31/12/2017 mil EUR	Cons 31/12/2016 mil EUR	Cons 31/12/2015 mil EUR	Cons 31/12/2014 mil EUR	Cons 31/12/2013 mil EUR
	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR	12 meses Aprobado IFRS AR
Tipo de cambio: USD/EUR	0.87336	0.83382	0.94868	0.91853	0.82365	0.72511
Total activo	2,234,168,307	2,102,703,157	2,253,094,575	2,213,333,462	2,169,621,695	1,937,000,992
Fondos propios	169,649,696	164,988,802	173,207,548	181,425,564	164,712,875	138,103,839
Ingresos operacionales	47,006,090	43,077,644	49,168,031	54,284,930	50,384,622	47,501,995
Resultado antes impuestos	17,371,170	14,314,188	6,746,991	17,329,844	15,385,875	16,362,121
Resultado neto	13,122,264	9,904,948	3,269,141	13,866,080	12,111,846	12,906,969
Capit. bursátil	143,708,451	172,761,046	152,299,419	143,671,420	150,346,918	148,940,230
Empleados	235,000	229,000	241,000	264,000	266,000	263,000

Fuente: Base de datos Osiris, Bureau Van Dijk.

ANEXO II. EJEMPLO DE TABLAS UTILIZADAS PARA LA REALIZACIÓN DEL ANÁLISIS DEA

Tabla 11. Situación de la DMU "CAIXABANK" en el año 2018, antes de realizar el análisis DEA

2018

EMPRESA	INPUT 1 (i+d+i)(MM€)	INPUT 2 (nºempleados)	OUTPUT 1 (BAI)(MM€)		PESOS (λ)
SANTANDER	1468	202713	14201		1
BBVA	1130	125627	8446		1
ING	1156	54000	6838		1
CAIXABANK	253	32552	2807		1
SOCIETE GENERALE	800	149022	6117		1
CREDIT AGRICOLE	695	73346	6496		1
LLOYDS BANK	1306,4	72626	6608		1
HSBC	1200	235000	17371		1
BNP PARIBAS	700	197162	10208		1
BARCLAYS	458	83500	3874		1
				EFICIENCIA	1

RESTRICCIONES

	LHS		RHS
INPUT 1	9166,4	<=	253
INPUT 2	1225548	<=	32552
OUTPUT 1	82966	>=	2807
Σλ	10	=	1

Fuente: Elaboración propia.

Tabla 12. Situación de la DMU "CAIXABANK" en el año 2018, después de realizar el análisis DEA

2018

EMPRESA	INPUT 1 (i+d+i)(MM€)	INPUT 2 (nºempleados)	OUTPUT 1 (BAI)(MM€)		PESOS (λ)
SANTANDER	1468	202713	14201		0
BBVA	1130	125627	8446		0
ING	1156	54000	6838		0
CAIXABANK	253	32552	2807		1
SOCIETE GENERALE	800	149022	6117		0
CREDIT AGRICOLE	695	73346	6496		0
LLOYDS BANK	1306,4	72626	6608		0
HSBC	1200	235000	17371		0
BNP PARIBAS	700	197162	10208		0
BARCLAYS	458	83500	3874		0
				EFICIENCIA	1

RESTRICCIONES

	LHS		RHS
INPUT 1	253	<=	253
INPUT 2	32552	<=	32552
OUTPUT 1	2807	>=	2807
Σλ	1	=	1

Fuente: Elaboración propia.

ANEXO III. DATOS ANUALES RECOGIDOS Y RESULTADOS DE EFICIENCIA POR EMPRESA

Tabla 13. Datos y resultados de eficiencia, años 2017-2018

	2018				2017			
	i+d+i (MM€)	nºemp	BAI (MM€)	EFICIENCIA	i+d+i (MM€)	nºemp	BAI (MM€)	EFICIENCIA
SANTANDER	1468	202713	14201	0,8905519	1470	195732	12091	0,8838633
BBVA	1130	125627	8446	0,7736982	693,25	131856	6931	0,7632454
ING	1156	54000	6838	1,0000000	1100	51504	7268	1,0000000
CAIXABANK	253	32552	2807	1,0000000	94	36972	2098	1,0000000
SOCIETE GENERALE	800	149022	6117	0,5798442	650	147125	5138	0,5363407
CREDIT AGRICOLE	695	73346	6496	0,7851778	614	73707	5929	0,6378889
LLOYDS BANK	1306,4	72626	6608	1,0000000	1309,4	75944	5942	1,0000000
HSBC	1200	235000	17371	1,0000000	880	229000	14314	1,0000000
BNP PARIBAS	700	197162	10208	1,0000000	900	196128	11310	0,8756245
BARCLAYS	458	83500	3874	0,6931089	412	79900	3989	0,7294780

Fuente: Elaboración propia.

Tabla 14. Datos y resultados de eficiencia, años 2015-2016

	2016				2015			
	i+d+i (MM€)	nºemp	BAI (MM€)	EFICIENCIA	i+d+i (MM€)	nºemp	BAI (MM€)	EFICIENCIA
SANTANDER	1327	191635	10768	0,9439330	1377	189464	9547	0,6203033
BBVA	693,25	134792	6392	0,7265756	693,25	137968	4602	0,5118061
ING	761	54000	5903	0,7534570	691	54000	6172	0,8106568
CAIXABANK	84,1	32403	1538	1,0000000	64,2	32242	638	1,0000000
SOCIETE GENERALE	650	145672	6307	0,6998765	540	145703	6109	0,6810545
CREDIT AGRICOLE	356	70830	3347	0,6126115	409	71495	3811	0,4495616
LLOYDS BANK	1294,8	80418	4946	0,4906458	1329,7	85703	2237	1,0000000
HSBC	835	241000	6746	1,0000000	800	264000	17329	1,0000000
BNP PARIBAS	684	192419	11210	1,0000000	650	189077	10379	0,8440219
BARCLAYS	389	119300	3769	0,5783756	365	129400	1559	0,3304217

Fuente: Elaboración propia.